VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

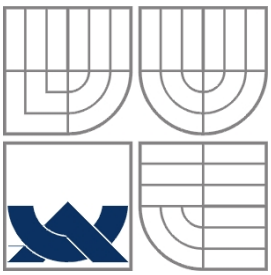
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SPOLEČNOST ZABÝVAJÍCÍ SE TELEFONOVÁNÍM

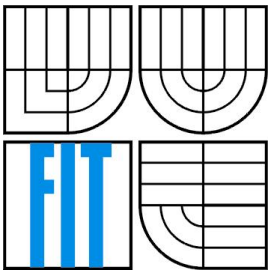
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
BC. MICHAL HUVAR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SPOLEČNOST ZABÝVAJÍCÍ SE TELEFONOVÁNÍM

SYSTEM FOR COMPANY DEALING WITH TELEPHONING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. MICHAL HUVAR

VEDOUCÍ PRÁCE
SUPERVISOR

MGR. ROMAN TRCHALÍK

BRNO 2011

Abstrakt

Cílem této diplomové práce je navrhnout a vytvořit systém pro společnost zabývající se telefonováním, který co nejvíce usnadní a automatizuje práci jejích zaměstnanců. Systém bude splňovat požadavky na moderní informační systém takovéto společnosti a bude propojen s telefonní ústřednou, ze které bude získávat informace o všech provedených hovorech včetně jejich hlasových záznamů. Práce je zaměřena na použití moderních kvalitních technologií a jejich integraci do jednoho systému. První část práce bude podrobně popisovat požadavky na tento systém a následné vytvoření návrhu. V druhé části budou uvedeny použité technologie a implementace jednotlivých částí včetně konfigurace a propojení s telefonní ústřednou. Nebude zde také chybět popis procesu měření výkonnosti a následné optimalizace kritických částí, u kterých je požadován vysoký výkon.

Abstract

The purpose of this master‘s thesis is to design and create system for company dealing with telephoning, which will simplify and automate the work of their employees as much as possible.
System will meet the requirements on modern information system of telemarketing company and will be connected to telephone private branch exchange, from which it will be getting information about all phone calls including their voice records. The work is focused on usage of modern quality technologies and their integration into one system. First part of this work will closely describe requirements on this system and resulting creation of design. In the second part there will be presented used technologies and implementation of particular sections including configuration and connection with telephone private branch exchange. There will be also description of performance measurement process and resulting optimalization of critical sections in which high performance is required.

Klíčová slova

telemarketing, informační systém, databáze, UML, Java EE, aplikační rámec Spring, Spring Web MVC, Hibernate, PostgreSQL, SQL, Javascript, DWR, Asterisk, CDR

Keywords

telemarketing, information system, database, UML, Java EE, Spring Framework, Spring Web MVC, Hibernate, PostgreSQL, SQL, Javascript, DWR, Asterisk, CDR

Citace

Huvar Michal: Systém pro společnost zabývající se telefonováním, diplomová práce, Brno, FIT VUT v Brně, 2011

Systém pro společnost zabývající se telefonováním

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Mgr. Romana Trchalíka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Huvar
20.5.2011

Poděkování

Chtěl bych velmi poděkovat Mgr. Romanu Trchalíkovi za poskytnutí podrobných požadavků na systém a za odborné vedení a konzultace, které mi poskytoval v průběhu vypracovávání této diplomové práce.

Obsah

1 Úvod

V dnešní době masivního rozvoje informačních technologií každá společnost disponuje moderním počítačovým vybavením. Logickým důsledkem je rostoucí poptávka po softwaru, který usnadní a zefektivní pracovní výsledky. Stejně tomu tak je i u společností zabývajících se telefonováním, které jsou dnes většinou nazývány „telemarketingové společnosti“. Tyto společnosti se začaly mohutně rozvíjet koncem 70. let minulého století, kdy se ještě veškerá administrativa provozovala papírovou formou. Koncem 90. let se však osobní počítače začínaly stávat čím dál tím dostupnějšími, a tak jimi tyto společnosti začínaly vybavovat i svá telefonní centra.

Cílem této diplomové práce je navrhnout informační systém právě pro takovouto společnost zabývající se telemarketingem, která má velkou část svých operátorů vybavenou osobním počítačem, ale stále ještě zaměstnává některé operátory pracující bez počítačů. Systém bude vytvořen za pomoci moderních technologií, bude spravovat veškeré projekty a zaměstnance společnosti a bude vybaven inteligentním algoritmem pro vybírání telefonních kontaktů pro jednotlivé operátory. Důležitým požadavkem moderní telemarketingové společnosti je monitorování provedených hovorů, které slouží jako kontrolní mechanismus pro práci jednotlivých operátorů. Zároveň ale také slouží ke sledování reakcí zákazníků a tím pádem k přizpůsobení formulace otázek kladených operátory pro získání lepších výsledků. Tento požadavek je také reflektován, a proto bude systém propojen s telefonní ústřednou, což umožní zobrazování informací o všech provedených hovorech včetně přehrávání jejich záznamů a zobrazování statistik jednotlivých operátorů. Výsledkem bude výrazné zjednodušení a urychlení práce zaměstnanců společnosti a tím pádem také její vyšší zisky.

První kapitola je úvodní a právě ji čtete. Druhá kapitola obsahuje stručný popis zadání vyvíjeného systému.

Ve třetí kapitole je podrobně rozebrána analýza požadavků. Přestože tato aplikace je vytvořena pouze v rámci školního studia jako diplomový projekt, který nebude pravděpodobně nikdy použit, chtěl jsem, aby dostala reálné rozměry. Kontaktoval jsem tedy osobu pracující v podobné telemarketingové společnosti a zjistil jsem od ní základní informace o tom, jak taková společnost funguje a jaké jsou kladeny nároky na její informační systém. Na základě takto získaných informací jsem sestavil patnáct požadavků, které jsou rozděleny do pěti logických kategorií. U každého požadavku je také uvedena předpokládaná a skutečná obtížnost jeho realizace.

Čtvrtá kapitola popisuje návrh řešení celého systému pomocí diagramů vytvořených v modelovacích nástrojích. Jsou zde uvedeny diagramy případů užití, konceptuální diagram tříd, návrh schématu databáze a návrh architektury aplikace.

Nejobsáhlejší kapitolou celé práce je kapitola pátá, která popisuje samotnou implementaci systému. Nejprve v ní jsou představeny vybrané technologie a nástroje použité k implementaci. Dále je zde stručně popsána konfigurace aplikace a následně je podrobněji rozebráno, jakým způsobem jsou implementovány jednotlivé vrstvy použité architektury. V této kapitole je také uvedena konfigurace telefonní ústředny Asterisk a její propojení s vyvíjeným systémem. V závěru této kapitoly je popsáno měření výkonnosti kritických částí systému a postup při optimalizaci na základě získaných výsledků pro zvýšení odezvy.

Poslední kapitolou je závěr, který tvoří shrnutí celé práce a naznačuje možná vylepšení či rozšíření do budoucna.

2 Informační systém telemarketingové společnosti

Telemarketingová společnost se zabývá aktivním telemarketingem a poskytuje dva typy služeb. Prvním z nich je nabízení produktů po telefonu a druhým je získávání informací z průzkumů. Společnost je tvořena různými odděleními, dceřinými společnostmi nebo pobočkami. Pro tyto pobočky budou realizovány jednotlivé marketingové kampaně. V textu jsou dále tyto pobočky nazývány klienty. Informační systém pro tuto společnost bude umožňovat správu všech zaměstnanců, klientů a telefonních seznamů s obvolávanými kontakty. Bude v něm možnost vytvářet a spravovat projekty pro jednotlivé klienty a vytvářet formuláře hovoru pro průzkumy a nabízení produktů Operátoři budou tyto formuláře vyplňovat na základě získaných informací z telefonních hovorů a systém je bude všechny evidovat pro pozdější zpracování. Společnost zaměstnává jak operátory pracující s PC, tak operátory bez počítače. Tuto skutečnost musí systém reflektovat a musí tedy umožňovat tisk nevyplněných formulářů hovorů a jejich zpětné zadávání do systému. Jako v každé jiné společnosti, tak i v této funguje určitá hierarchie pracovníků. Jsou zde operátoři, manažeři a vedoucí pracovníci. Každý pracovník tedy musí mít přístup pouze k informacím, ke kterým má oprávnění. Systém bude propojen s telefonní ústřednou, na kterou budou směrovány všechny hovory a ze které bude sbírat technické informace a záznamy o provedených hovorech. Ty potom budou dostupné vedoucím pracovníkům za účelem sbírání statistik, kontroly práce operátorů a zkvalitňování služeb.

Uživatelské rozhraní bude implementováno prostřednictvím HTML stránek a bude se s ním tedy pracovat pomocí webového prohlížeče. K systému se bude přistupovat pouze přes vnitřní síť společnosti a nebude tedy přístupný přes internetovou síť. Z tohoto důvodu nejsou kladeny velké požadavky na jeho bezpečnost proti různým síťovým útokům, ale pouze proti neoprávněnému přístupu jednotlivých uživatelů. Naopak velký důraz je kladen na výkonnost, jelikož je požadována rychlá odezva i při vysokém zatížení.

3 Analýza a sběr požadavků

V této kapitole budou definovány základní požadavky na vytvářený systém, které byly vytvořeny na základě důkladné analýzy potenciálních požadavků fiktivní moderní telemarketingové společnosti. Každý požadavek je charakterizován několika body:

- Pořadové číslo požadavku v rámci analýzy a jeho název
- Zadání požadavku
- Odhadovaná náročnost při řešení (1 – nejnižší… 5 - nejvyšší)
- Skutečná náročnost zjištěná zpětně po implementaci (1 – nejnižší… 5 - nejvyšší)

Náročnost je myšlena pouze v rámci implementace tohoto systému, nikoliv obecně. Nejvyšší stupeň tedy reprezentuje požadavky, které jsou nejsložitější z implementace celého systému. Porovnání odhadované a skutečné náročnosti poslouží velmi dobře jako zpětná vazba pro zlepšení analytického myšlení a vytváření časových odhadů na realizaci různých úkolů. Zároveň však může být zajímavé pro čtenáře, který tak snadno zjistí, jaké části systému byly nejnáročnější a jaké části byly z hlediska náročnosti podceněny nebo naopak přeceněny.

3.1 Uživatelé a subjekty v systému

V systému budou figurovat celkem dva subjekty. Prvním je samotná společnost, která je tvořena zaměstnanci neboli uživateli systému a jednotlivými klienty. Druhým subjektem jsou telefonní kontakty, které se v rámci těchto kampaní obvolávají. V případě, že se kampaní rozumí nabízení nějakého produktu, jsou to vlastně potenciální zákazníci – kupci daného produktu. Klíčovým subjektem jsou uživatelé systému, kteří budou dále detailně popsáni. Na základě analýzy požadavků bylo stanoveno celkem pět uživatelských rolí.

Uživatel

Uživatelé s touto rolí nebudou mít přístup do žádné sekce systému. Budou si moci pouze změnit svoje osobní údaje. Jelikož systém by měl sloužit také jako databáze všech zaměstnanců společnosti, bude role uživatel přidělována zaměstnancům, kteří nebudou přímo pracovat s tímto systémem, a bude zde tedy pouze z důvodu evidence zaměstnanců.

Operátor bez PC

Operátoři, kteří nebudou mít k dispozici osobní počítač, nebudou mít ani přístup do systému. Z toho důvodu bude mít tato role stejná přístupová práva jako role uživatel. Bude se ale využívat při zadávání vytištěných formulářů hovoru, které vyplnili právě tito operátoři, aby systém u každého uloženého hovoru evidoval operátora, který jej provedl. Uživatelé s rolí operátor bez PC budou v textu nadále zmiňováni pod pojmem „operátor“.

Operátor s PC

Roli operátora s PC bude mít patrně nejvíce uživatelů v systému. Bude se jednat o zaměstnance provádějící telefonní hovory. Budou mít stejná práva jako role uživatel s tím, že navíc budou moci přistoupit do sekce telefonního hovoru, která bude blíže rozebrána v následujících požadavcích. Uživatelé s rolí operátor s PC budou v textu nadále zmiňováni pod pojmem „operátor s PC“.

Manažer

Uživatelé s rolí manažera budou vedoucí pracovníci společnosti, kteří budou mít na starosti jednotlivé projekty klientů a telefonní operátory. Budou mít oprávnění k přístupu do sekcí uživatelů, klientů, projektů a telefonních hovorů. K projektům budou moci vytvářet a spravovat nabízené produkty a průzkumy. Vždy však budou mít přístup pouze k datům, která se týkají projektů, k nimž jsou přiřazeni. Uživatelé s touto rolí budou v textu nadále zmiňováni pod pojmem „manažer“.

Administrátor

Roli administrátor budou mít nejvýše postavení pracovníci, kteří tak budou mít neomezený přístup do všech částí systému. Oproti uživatelům s rolí manažer budou moci přidávat/odebírat uživatele, přidávat klienty a vytvářet projekty. Uživatelé s rolí administrátor budou v textu nadále zmiňováni pod pojmem „administrátor“. Uživatelem se v textu bude nadále myslet běžný uživatel, operátor, operátor s PC, manažer i administrátor.

Požadavek č. 1 – uživatelské role

Systém musí rozlišovat uživatele na základě pěti rolí definovaných v předchozí kapitole a jednotlivé části zobrazovat pouze oprávněným uživatelům. Každý uživatel systému musí mít přidělenou právě jednu roli. Přístup do systému bude umožněn pouze po přihlášení, které se bude sestávat z procesu autentizace a autorizace. Autentizace je proces ověřování proklamované identity subjektu, kdy uživatel zadá svoje uživatelské jméno a heslo a systém ověří, zda je nějaký uživatel s takovým uživatelským jménem a heslem zaregistrován a povolen. Po úspěšné autentizaci bude následovat proces autorizace, který na základě získané identity uděluje oprávnění nebo odepření přístupu do dané části systému. Po úspěšném přihlášení si bude systém identitu uživatele uchovávat a nebude nutné provádět proces autentizace znovu. Proces autorizace se však bude provádět vždy znovu. Pokud uživatel neprovede žádnou akci po dobu 15 minut, bude automaticky odhlášen.
Náročnost: 3
Skutečná náročnost: 2

Požadavek č. 2 – správa uživatelů

Systém musí umožňovat kompletní správu uživatelů. Manažeři si budou moci zobrazovat seznam všech uživatelů a jejich detailní informace. Administrátoři budou moci navíc přidávat, editovat a odebírat uživatele. Odebrání uživatele nepovede k jeho smazání ze systému, ale pouze k jeho deaktivaci. Neaktivní uživatel se nebude moci přihlásit a nebude možné s ním pracovat v žádných projektech. Povinné údaje při registraci nového uživatele budou:

- Jméno – řetězec dlouhý maximálně 255 znaků
- Příjmení – řetězec dlouhý maximálně 255 znaků
- Heslo – řetězec dlouhý maximálně 255 znaků pro přihlašovací heslo do systému
- Pracovní poměr – výběr ze čtyř předdefinovaných hodnot – zaměstnanec, brigádník, dohoda o pracovní činnosti a dohoda o provedení práce
- Role – výběr z pěti uživatelských rolí definovaných v kapitole 3.1
- Ulice – řetězec dlouhý maximálně 255 znaků
- Město – řetězec dlouhý maximálně 255 znaků
- PSČ – celé číslo
- Okres – řetězec dlouhý maximálně 255 znaků

- Region – výběr jednoho ze čtrnácti regionů České republiky
- Pracovní doba – sedm řetězců dlouhých maximálně 50 znaků reprezentujících každý den v týdnu

Pokud bude vybrána role operátora s PC, bude k povinným údajům patřit také:

- Identifikátor v telefonní ústředně – řetězec dlouhý maximálně 255 znaků, pod kterým bude uživatel zaregistrován v telefonní ústředně
- Heslo v telefonní ústředně – řetězec dlouhý maximálně 255 znaků

Nepovinné údaje budou:

- Titul – řetězec dlouhý maximálně 50 znaků
- Datum narození – datum bez časové zóny
- E-mail – řetězec maximálně 255 znaků dlouhý

Uživatelské jméno, které bude sloužit k identifikaci uživatele, nepůjde libovolně zvolit. Systém ho bude generovat automaticky z příjmení uživatele a zaručí jeho jedinečnost v rámci celého systému. Po úspěšném přihlášení si bude každý uživatel moci změnit nastavení svých osobních údajů kromě pracovního poměru, role, údajů k připojení k telefonní ústředně a pracovní doby.
Náročnost: 2
Skutečná náročnost: 2

Požadavek č. 3 – správa klientů

Klienty budou moci do systému vkládat a editovat pouze administrátoři. Ti si budou moci také zobrazovat seznam všech klientů a jejich detailní informace. Manažeři si budou moci zobrazovat informace pouze o těch klientech, k jejichž projektům jsou přiřazeni. Povinné údaje při přidávání nového klienta budou:

- Název klienta – řetězec maximálně 255 znaků dlouhý
- Jméno kontaktní osoby – řetězec maximálně 255 znaků dlouhý
- Příjmení kontaktní osoby – řetězec maximálně 255 znaků dlouhý
- Funkce kontaktní osoby – řetězec maximálně 255 znaků dlouhý reprezentující funkci kontaktní osoby v dané pobočce
- Telefon kontaktní osoby – řetězec maximálně 255 znaků dlouhý
- Ulice – řetězec dlouhý maximálně 255 znaků
- Město – řetězec dlouhý maximálně 255 znaků
- PSČ – celé číslo
- Okres – řetězec dlouhý maximálně 255 znaků
- Region – výběr jednoho ze čtrnácti regionů České republiky

Nepovinné údaje budou:

- IČ – řetězec dlouhý maximálně 10 znaků reprezentující identifikační číslo subjektu
- DIČ – řetězec dlouhý maximálně 20 znaků reprezentující daňové identifikační číslo subjektu
- Právní forma – řetězec dlouhý maximálně 255 znaků určující typ podnikatelského subjektu
- Poznámky – řetězec s neomezenou délkou

Náročnost: 2

Skutečná náročnost: 1

Požadavek č. 4 – import telefonních kontaktů

Pro každého klienta bude možné importovat neomezené množství seznamů telefonních kontaktů. Tyto seznamy budou sloužit jako zdroj telefonních čísel, která se budou obvolávat v rámci klientových projektů. Importovaný seznam bude muset být ve formátu CSV, kde jednotlivé položky budou odděleny čárkou a budou mít následující strukturu:

- Telefonní číslo – řetězec dlouhý maximálně 255 znaků jednoznačně identifikující daný kontakt
- Titul – řetězec dlouhý maximálně 50 znaků
- Jméno – řetězec dlouhý maximálně 255 znaků
- Příjmení – řetězec dlouhý maximálně 255 znaků
- Datum narození – datum bez časové zóny
- Adresa – řetězec dlouhý maximálně 512 znaků obsahující kompletní adresu včetně města a poštovního směrovacího čísla
- E-mail – řetězec dlouhý maximálně 255 znaků
- Společnost – řetězec dlouhý maximálně 255 znaků reprezentující název firmy či zaměstnavatele

Jedinou povinnou položkou bude telefonní číslo. Pokud bude některá další položka chybět, nebude v seznamu uvedena, ovšem oddělovací čárka musí být uvedena vždy. Pokud bude vyplněno datum narození, tak se vypočítá věk dané osoby z data narození a aktuálního data. Následně se uloží spolu s ostatními údaji kontaktu. Údaj o věku bude sloužit zejména operátorům, kteří tak rychle uvidí, s osobou jaké věkové kategorie vedou hovor a nebudou se muset zatěžovat počítáním věku z data narození. Při importu se bude také kontrolovat duplicita kontaktů. Pokud již importovaný kontakt v systému bude existovat, nebude se importovat znovu, ale pouze se přiřadí také aktuálně importovanému seznamu. V systému tedy budou moci být kontakty, které patří do více seznamů. Oprávnění importovat telefonní seznam musí mít pouze administrátoři a manažeři, kteří jsou přiřazeni k alespoň jednomu projektu daného klienta.

Ke každému seznamu bude možno zobrazit všechny jeho telefonní kontakty. Ke každému kontaktu půjde zobrazit všechny jeho údaje a bude možné jej přidat na černou listinu či odebrat z této listiny. Přidání na černou listinu bude mít za důsledek, že daný kontakt nebude aktivní a nebude nadále vybírán pro provedení telefonních hovorů. Seznam kontaktů může obsahovat desítky tisíc položek, a proto je nutné při jeho prohlížení zobrazovat vždy jen určité množství kontaktů na jednu stránku. Implicitně to bude 20 kontaktů na stránku. Toto množství by však mělo být nastavitelné v rozsahu od 20 do 500 položek. Pro usnadnění vyhledávání v seznamu musí být možnost filtrování zobrazených kontaktů alespoň podle telefonního čísla, jména, příjmení a e-mailu.
Náročnost: 4
Skutečná náročnost: 3

3.2 Projekty

Projektem v systému se rozumí kampaň objednaná klientem. Veškeré nabízení produktů a provádění průzkumů tak bude součástí nějakého projektu.

Požadavek č. 5 – správa projektů

Ke každému klientovi bude možné vytvořit neomezené množství projektů. Oprávnění pro tuto funkcionalitu budou mít pouze administrátoři, kteří budou moci také všechny projekty editovat. Každý projekt bude muset mít vybraného alespoň jednoho manažera a vyplněny následující údaje:

- Název – řetězec dlouhý maximálně 255 znaků
- Datum zahájení – datum bez časové zóny
- Datum ukončení – datum bez časové zóny

Manažerům, kteří jsou k projektu přiděleni, bude umožněn přístup do všech sekcí spojených s daným projektem a klientem. Ostatní manažeři tento přístup mít nebudou a neuvidí tedy žádné informace k tomuto projektu. Stejně jako seznam telefonních kontaktů i seznam projektů musí mít nastavitelný rozsah velikosti zobrazené stránky od 20 do 500 položek a musí být filtrovatelný alespoň podle názvu a klienta.
Náročnost: 3
Skutečná náročnost: 2

Požadavek č. 6 – správa produktů

Ke každému projektu bude možnost vytvořit nový produkt. Oprávnění na vytvoření a editaci produktu budou mít všichni administrátoři a manažeři, kteří jsou k danému projektu přiřazeni. Pouze tito uživatelé pak také budou moci spravovat daný produkt. Povinné údaje budou:

- Název – řetězec dlouhý maximálně 255 znaků
- Popis – řetězec s neomezenou délkou
- Cena – číslo s desetinnou čárkou
- Datum zahájení – datum bez časové zóny reprezentující datum zahájení nabízení produktu

Povinně se bude muset také vybrat jeden ze seznamů telefonních kontaktů klienta, který se použije pro daný produkt. Nepovinné údaje budou:

- Datum ukončení – datum bez časové zóny
- Internetové stránky – řetězec dlouhý maximálně 512 znaků, který bude obsahovat oficiální internetové stránky daného produktu

K vytvořenému produktu bude možné přidávat a odebírat fotografie, operátory a také vytvářet formuláře hovoru. Přiřazení operátoři potom tento produkt budou telefonicky nabízet potenciálním zákazníkům.

Formuláře hovoru se budou skládat z otázek, které budou operátoři klást při nabízení daného produktu a z polí pro odpovědi, do kterých budou zapisovat reakce zákazníků. Do formuláře půjdou vložit tři typy polí – textové pole, výběrové pole a zaškrtávací pole. U textového pole bude možné zadat počet řádků, počet sloupců, název otázky, a zda se jedná o povinné pole. Počet řádků a sloupců bude určovat, jakou velikost bude mít textové pole pro odpověď. U zaškrtávacího pole půjde zadat pouze název otázky. U výběrového pole půjde zadat název otázky, zda se jedná o povinné pole a odpovědi, ze kterých bude možno vybírat. U všech tří polí půjde zvolit pořadí, ve kterém budou dané otázky ve formuláři zobrazeny. Implicitně se každá nová otázka zařadí nakonec formuláře. Výsledný formulář hovoru bude k nahlédnutí na detailu daného produktu a půjde kdykoliv editovat. Jednotlivé otázky půjdou mazat nebo jednoduše měnit jejich pořadí.

Náročnost: 4
Skutečná náročnost: 5

Požadavek č. 7 – vytváření průzkumů

Podobně jako vytváření produktů bude systém umožňovat také vytváření průzkumů. Tuto možnost budou mít opět pouze administrátoři a ti manažeři, kteří jsou k danému projektu přiřazeni. Průzkum se od produktu liší v tom, že jeho cílem je pouze sbírat informace od uživatelů, nikoliv je přesvědčit ke koupi určitého produktu. Povinnými údaji u průzkumu budou název, popis, datum zahájení průzkumu a seznam telefonních kontaktů klienta. Nepovinné bude datum ukončení průzkumu. Na všechny údaje se kladou stejné požadavky jako v případě produktu, proto zde již nejsou detailně popsány. K vytvořenému průzkumu bude možno přidávat operátory, kteří jej budou provádět a také vytvářet a editovat formulář hovoru naprosto stejným způsobem jako u produktu.
Náročnost: 4
Skutečná náročnost: 1

Požadavek č. 8 – spouštění kampaně

Kampaní se rozumí produkt nebo průzkum. Pokud tedy nebude potřeba přímo specifikovat, o kterou z těchto dvou kampaní se jedná, bude se nadále v textu používat pojmenování kampaň. Jakmile bude kampaň uložena, bude administrátorům a přiřazeným manažerům umožněno tuto kampaň spustit. Spuštění bude realizováno stiskem tlačítka na jejím detailu. Po stisku tohoto tlačítka se zkontroluje, zda má daná kampaň přidaného alespoň jednoho operátora pracujícího s PC, a zda má vytvořený formulář hovoru. Pokud ne, zobrazí se informace o tom, že je potřeba vybrat alespoň jednoho operátora nebo vytvořit formulář hovoru. Pokud bude vše v pořádku, zobrazí se dialogové okno pro potvrzení spuštění kampaně a po jeho potvrzení bude daná kampaň spuštěna. Podmínka existence operátora pracujícího s PC je z toho důvodu, aby mohl obvolávat zákazníky, kteří si přáli zavolat později. Operátor bez PC tuto možnost mít nebude. Přesnější popis funkcionality telefonních hovorů je popsán v následující kapitole požadavků. Po spuštění kampaně systém povolí přiřazeným operátorům nabízet daný produkt či provádět průzkum. Stejným způsobem jako spuštění kampaně bude umožněno i její zastavení. Zastavený produkt či průzkum se nebude přiřazeným operátorům nabízet. Zastavenou kampaň bude možno opět spustit.
Náročnost: 1
Skutečná náročnost: 1

3.3 Telefonní hovory operátorů pracujících na PC

Většinu uživatelů systému budou tvořit telefonní operátoři pracující na PC. Přestože budou mít velmi omezený přístup do systému, tak funkcionalita pro jejich obsloužení bude stěžejní. V následujících podkapitolách budou popsány požadavky na provádění telefonních hovorů pro operátory pracující na PC. Každý hovor bude mít v systému svoji reprezentaci. Tato reprezentace bude dále v textu zmiňována jako „**systémový hovor**“ a bude uchovávat kampaň, operátora, telefonní kontakt, vyplněný formulář hovoru a případně další atributy, které vyplynou z následujících požadavků.

Požadavek č. 9 – detail hovoru

Pokud bude operátor pracující na PC přiřazen k nějaké kampani a ta bude spuštěna, tak se mu po přihlášení vytvoří systémový hovor k této kampani a jeho detail se zobrazí. Funkcionalita výběru telefonního kontaktu pro tento hovor je popsána v požadavku č. 11. Pokud nebude operátor přirazen k žádné kampani nebo bude kampaň zastavena, bude o tom informován příslušnou hláškou a bude si pouze moci změnit své osobní údaje.

Detail hovoru bude obsahovat jednoznačné identifikační číslo v rámci dané kampaně a všechny důležité informace k úspěšnému provedení telefonního hovoru. Budou zde název kampaně, název klienta a v případě produktu také webové stránky, cena a všechny fotografie ve zmenšeném formátu. Po výběru některé fotografie se tato zobrazí v původní velikosti, ve které byla k produktu uložena. Dále zde budou informace o telefonním kontaktu, kterému se bude volat. Bude to titul, jméno, příjmení, věk a telefonní číslo. Další detailnější informace o dané kampani, klientovi a telefonním kontaktu si bude možné zobrazit v modálním okně. Při zobrazení tohoto okna se nesmí ztratit žádná operátorem vyplněná data. Nakonec se zobrazí formulář hovoru v takové podobě, v jaké byl vytvořen administrátorem či manažerem daného projektu. Kromě všech otázek seřazených ve správném pořadí se zobrazí zaškrtávací pole pro přidání daného telefonního kontaktu na černou listinu. Uložení hovoru s touto volbou bude mít stejný význam jako přidání kontaktu na černou listinu v seznamu telefonních kontaktů daného klienta a nebude tedy nadále vybírán pro žádný hovor. V případě, že se bude jednat o kampaň typu produkt, tak se také zobrazí výběrové pole pro volbu reakce zákazníka. Toto pole bude povinné a půjdou v něm zvolit tři možnosti – má zájem, nemá zájem a přeje si zavolat později. Pokud operátor zvolí možnost zavolání později, tak se zobrazí pole pro výběr data a času, do kterého bude muset povinně vyplnit, kdy si přeje zákazník znovu zavolat. Pod formulářem hovoru bude tlačítko pro spuštění hovoru.

Náročnost: 3

Skutečná náročnost: 3

Požadavek č. 10 – provedení telefonního hovoru

Každý počítač operátora bude vybaven softwarovým telefonem. Po stisku tlačítka na spuštění hovoru se systém spojí s telefonní ústřednou, přihlásí daného operátora a vytočí telefonní číslo kontaktu, kterému se má v tu chvíli volat. Tlačítko pro spuštění hovoru se v tu chvíli změní na tlačítko pro ukončení hovoru a zároveň se objeví tlačítko pro uložení hovoru. Po stisku tlačítka pro ukončení se ukončí spojení s daným telefonním číslem, pokud již nebylo ukončeno ze strany volaného a také s ústřednou, ze které je operátor odhlášen. Následně operátor vyplní formulář hovoru, pokud ho již nevyplnil v průběhu hovoru a bude moci systémový hovor uložit. Před uložením se zkontroluje, zda jsou vyplněny všechny povinné údaje formuláře. Pokud ne, tak se zobrazí informace o tom, které povinné údaje chybí a pole pro tyto údaje budou zvýrazněna, aby je operátor rychle našel a doplnil. Pokud bude vše v pořádku, vyplněný formulář se spolu s informacemi o daném hovoru uloží do systému a operátorovi se vytvoří nový systémový hovor pro další telefonní kontakt a zobrazí jeho detail. Systém tak bude uchovávat informace s vyplněným formulářem o každém provedeném hovoru ke všem produktům a průzkumům.

Systém musí nahrávat záznamy všech telefonních hovorů, které operátoři provedou, a musí o nich ukládat základní technické informace získané z telefonní ústředny. Mezi těmito informacemi by neměl chybět přesný čas a datum začátku hovoru, hrubá délka hovoru, čistá délka hovoru, operátorův identifikátor v telefonní ústředně a telefonní číslo volaného. Hrubá délka hovoru je čas od započetí

vytáčení po položení hovoru. Čistá délka je čas od přijetí hovoru ze strany volaného do položení hovoru.
Náročnost: 5
Skutečná náročnost: 5

Požadavek č. 11 – vybírání telefonních kontaktů

V této kapitole bude popsáno, jakým způsobem bude systém vybírat telefonní kontakt pro systémový hovor operátora. Pro každý hovor se bude vybírat z kontaktů telefonního seznamu, který je přiřazený k dané kampani. Nejprve se budou kontakty vybírat náhodně. Výjimku budou tvořit pouze ty kontakty, kterým se již v rámci dané kampaně volalo, ale přáli si zavolat později. Při výběru se tedy bude brát zřetel na to, zda existuje nějaký kontakt, kterému se má v danou dobu zavolat nebo kterému se již mělo zavolat. Pokud takový kontakt existuje, vybere se ten s nejstarším datem a časem pozdějšího zavolání. Po výběru kontaktu systém zajistí, aby nebyl vybrán znovu, dokud nebudou vybrány všechny ostatní kontakty v telefonním seznamu kampaně. Pokud již v seznamu kontaktů nebude žádný kontakt, ke kterému ještě nebyl proveden hovor, tak se budou vybírat kontakty od nejstaršího provedeného hovoru. Pořadí hovorů se tedy bude opakovat jako v předchozím kole a tím se zamezí, aby se některému zákazníkovi volalo v blízkém časovém intervalu po sobě. Celá tato funkcionalita musí být velmi rychlá, ideálně okamžitá, aby operátor neztrácel čas mezi jednotlivými hovory. Čekání déle jak 10 sekund je nepřípustné.

Jakmile je telefonní kontakt vybrán, vytvoří se systémový hovor a danému operátorovi se zobrazí detail hovoru popsaný v požadavku č. 9. Pokud operátor v tu chvíli daný hovor neprovede, ať už z důvodu poruchy systému či náhlého ukončení práce a odhlášení, zobrazí se mu při dalším přihlášení do systému právě tento nedokončený hovor. Nový systémový hovor s novým telefonním kontaktem se tedy vytvoří pouze tehdy, pokud byl u předchozího hovoru řádně vyplněn formulář hovoru. Z toho vyplývá, že nebude možné, aby operátor přeskakoval mezi jednotlivými hovory.
Náročnost: 4
Skutečná náročnost: 5

3.4 Telefonní hovory operátorů bez PC

Většina operátorů v marketingové společnosti bude mít přístup k PC. Přesto však společnost ještě bude zaměstnávat operátory, zejména brigádníky, kteří nebudou mít přístup k PC, ale budou mít pouze telefon. Systém pro to musí být náležitě uzpůsoben. V následujících podkapitolách budou definovány požadavky na správu telefonních hovorů provedených operátory bez PC.

Požadavek č. 12 – tisk formulářů hovoru

Administrátorům a manažerům přiřazeným k danému projektu bude umožněno vytvářet sady formulářů hovoru pro danou kampaň pro operátory pracující bez PC. Tato sada bude vlastně seznam systémových hovorů, které se vytisknou a předají operátorovi bez PC. Ten je po provedení telefonních hovorů vyplní a manažer projektu je následně manuálně zadá do systému. Při vytváření nové sady se zadá počet hovorů, který má být v dané sadě vygenerován a systém automaticky vybere telefonní kontakty, vytvoří z nich systémové hovory a uloží danou sadu. Vybírání telefonních kontaktů bude probíhat stejným způsobem jako pro operátory pracující na PC s tím rozdílem, že se nebude brát ohled na ty kontakty, kterým se má zvolat později. Tyto kontakty budou zpracovány

operátory s PC, jelikož podle požadavku č. 8 musí mít každá kampaň přiřazeného alespoň jednoho operátora s PC.

Po úspěšném vytvoření nové sady formulářů hovoru se zobrazí detail dané sady, na kterém bude uveden název kampaně, projektu a klienta s odkazy na jejich detailní informace. Dále tam také bude seznam vygenerovaných systémových hovoru a tlačítko pro tisk. Po stisku tlačítka pro tisk se daná sada převede do formátu PDF, ze kterého půjde na tiskárně vytisknout. Ve výsledném PDF bude každý hovor na nové stránce a bude obsahovat hlavičku s ID hovoru, telefonním číslem, jménem, příjmením, věkem telefonního kontaktu a formulář hovoru. Formulář hovoru bude odpovídat elektronické podobě, která je zobrazena operátorům na PC. Textové pole se zobrazí jako rámeček příslušné velikosti, do kterého se bude vypisovat odpověď. Výběrové pole se zobrazí jako číslovaný seznam jednotlivých možností výběru, které se budou zaškrtávat. Zaškrtávací pole se zobrazí jako čtvereček, který se zaškrtne nebo ponechá prázdný. Každé povinné pole bude označeno vykřičníkem a daná otázka bude mít červenou barvu.
Náročnost: 3
Skutečná náročnost: 3

Požadavek č. 13 – zpracování formulářů hovoru

Manažerům musí být umožněno zadávat vytištěné vyplněné formuláře hovoru do systému. Z detailu dané kampaně si bude možné zobrazit seznam všech sad formulářů hovoru vytvořených k této kampani. Seznam bude obsahovat filtr pro snadné vyhledání sady formulářů, které bude manažer zadávat do systému. Filtr bude obsahovat zaškrtávací pole pro zatím nevyplněné sady, které bude implicitně zaškrtnuto. Dále bude umožňovat zvolit datum, po kterém byla sada vytvořena a identifikační číslo hovoru od a do. V případě vyplnění identifikačního čísla od nebo do se zobrazí sady obsahující všechny systémové hovory s identifikačními čísly v zadaném rozmezí.

Po výběru některé sady se zobrazí detail dané sady obsahující seznam všech jejích systémových hovorů. Manažer tak bude moci vybrat hovor, který začne vyplňovat. Typicky to bude první hovor dané sady. Vyplnění formuláře bude probíhat stejně jako by jej vyplňoval operátor pracující na PC s tím rozdílem, že zde nebude možnost zvolit zavolání později a bude zde nutné povinně vybrat jednoho z operátorů přiřazených k projektu, který daný hovor provedl. Pod formulářem budou dvě tlačítka *Uložit a skončit* a *Uložit a vyplnit další*. Po výběru druhého z nich se zobrazí další formulář pro hovor s následujícím identifikačním číslem v dané sadě, který ještě nebyl vyplněn. Toto tlačítko bude urychlovat zadávání vytištěných vyplněných formulářů, jelikož se bude předpokládat, že je bude manažer zadávat v pořadí, v jakém byla vytvořena. Pokud již žádný další nevyplněný hovor v sadě není, zobrazí se detail dané sady s informační hláškou, že je již celá sada vyplněna. Vyplněné formuláře hovoru půjde kdykoliv editovat.

Pokud operátor nestihl do konce své směny provést všechny telefonní hovory a vyplnit všechny formuláře, bude je moci obvolat v další směně nebo je může manažer smazat v seznamu systémových hovorů dané sady. Smazat ovšem půjdou pouze ty hovory, které ještě nebyly vyplněny. Smazáním systémového hovoru z dané sady se uvolní telefonní kontakt tohoto hovoru a systém ho bude moci opět vybrat pro nově vytvořené systémové hovory jak pro operátory s PC, tak pro ty bez PC.
Náročnost: 3
Skutečná náročnost: 3

3.5 Zpracování dat a statistiky

Systém musí umožňovat zobrazovat všechna data získaná z uskutečněných telefonních hovorů a tato data zpracovávat a odesílat klientovi nebo tisknout. Na základě zpracovaných dat potom jednotliví klienti budou moci vyhodnocovat průběh svých kampaní. V případě nabízení produktů budou zákazníkům, kteří měli zájem, zprostředkovávat dané produkty. V případě průzkumů budou ze získaných odpovědí vytvářet statistiky a výsledky průzkumů.

Požadavek č. 14 – zobrazení statistik

Administrátoři a manažeři přiřazení k danému projektu si budou moci na detailu každé kampaně zobrazit statistiku hovorů. Tato statistika bude reprezentována seznamem všech provedených hovorů, číslem udávajícím jejich počet a průměrnou čistou dobou hovoru. Seznam bude obsahovat filtr podle identifikačního čísla hovoru, telefonního čísla a jména kontaktu, operátora, který hovor provedl, data od, data do a v případě produktu také podle toho, zda měl volaný zájem nebo ne. Pokud bude zadáno datum od a datum do, tak se zobrazí pouze ty hovory, které byly provedeny v rozmezí těchto dvou dat. Administrátor si bude moci navíc filtrovat podle produktu nebo průzkumu. Seznam bude obsahovat datum hovoru včetně času, ID hovoru, operátora, telefonní číslo, čistou délku hovoru a možnost zobrazení detailu. Seznam bude moci být podle všech těchto atributů seřazen vzestupně i sestupně a implicitně bude seřazen podle data hovoru sestupně.

Po výběru některého detailu hovoru se zobrazí všechny informace, které jsou o daném hovoru v systému dostupné včetně vyplněného formuláře hovoru a dat z telefonní ústředny. Bude zde také možnost přehrát uložený záznam hovoru.

Náročnost: 3

Skutečná náročnost: 4

Požadavek č. 15 – zpracování dat

Na detailu každého produktu bude tlačítko s názvem „Zpracovat data“, které bude dostupné pouze pro administrátory. Po jeho stisknutí se zobrazí formulář s volbami pro zpracování dat daného produktu, které umožní blíže specifikovat, jaké hovory mají být zpracovány. Mezi těmito volbami bude výběr data od kdy a do kdy byly hovory uskutečněny, zda mají být vybrány pouze hovory, kdy měl zákazník zájem nebo zda mají být vybrány doposud nezpracované hovory. Dále bude formulář obsahovat výběrové pole s dvěma možnostmi, jak předat zpracovaná data klientovi, a to buď zasláním e-mailem, nebo vytištěním. Implicitně bude zvoleno vytištění. Pokud uživatel vybere volbu e-mailem, tak se ještě ve formuláři zobrazí pole pro zadání e-mailu, na který systém odešle zpracovaná data. Posledním prvkem zde bude tlačítko s názvem „Zpracovat“, po jehož stisku se v závislosti na zvoleném nastavení vyberou uložené systémové hovory daného produktu a zpracují se zadaným způsobem. Stejným způsobem půjdou zpracovávat data i ke každému průzkumu s tím rozdílem, že zde bude chybět volba, zda měl zákazník zájem, jelikož tato informace se ve formulářích hovoru průzkumu nezobrazuje.

Náročnost: 5

Skutečná náročnost:

4 Návrh systému

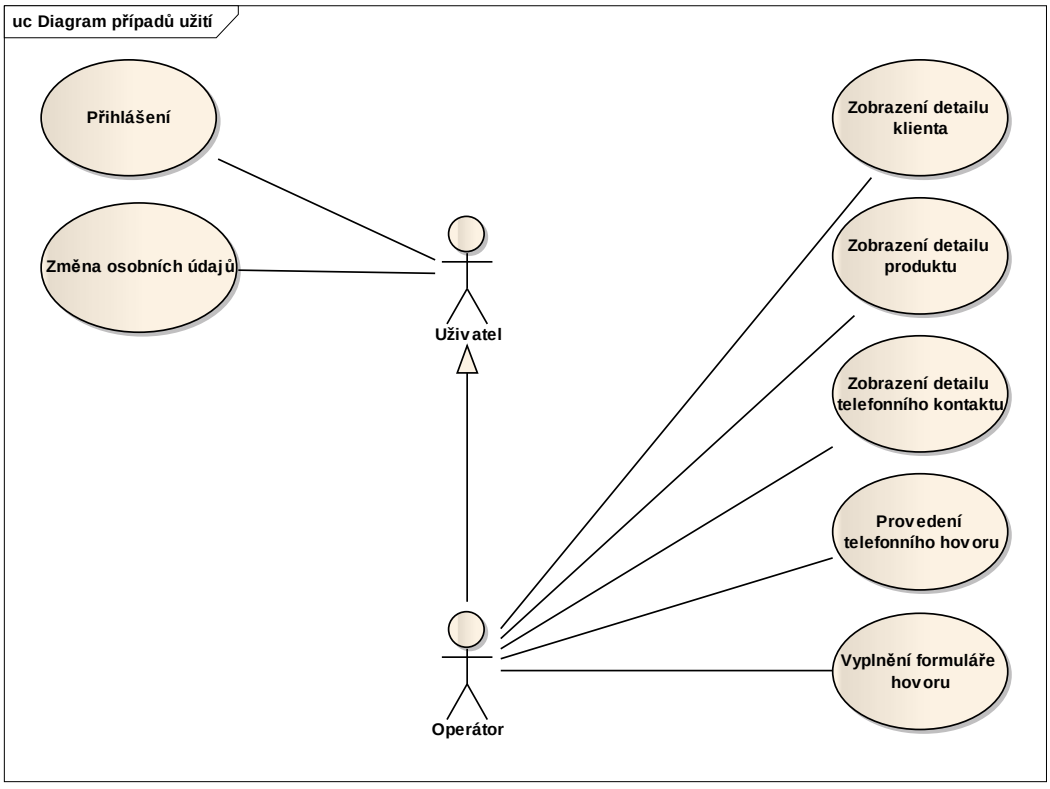
V této kapitole bude podrobně popsán návrh systému na základě výše popsaných požadavků. Návrh bude vymodelován v jazyce UML [1], což je vizuální modelovaný jazyk doplněný textem. Chování systému bude vymodelováno pomocí diagramů případů užití. Logický pohled na statickou strukturu systému bude vymodelován v konceptuálním diagramu tříd a v návrhu schématu databáze. V závěru kapitoly bude uveden a popsán návrh zvolené architektury pro implementaci systému s úrovněmi hierarchie objektů a omezením komunikace mezi nimi. Pro tvorbu všech modelů jsem použil modelovací nástroj Enterprise Architect, který je volně stažitelný v 30 denní zkušební verzi z webových stránek [2].

4.1 Diagramy případů užití

Diagramy případů užití jsou rozděleny podle aktérů na tři diagramy. Všichni aktéři na jednotlivých diagramech reprezentují uživatelské role v systému. Základním aktérem je *Uživatel*, ze kterého vycházejí všichni ostatní aktéři. Každý diagram případů užití by měl být doplněn textovým popisem. Tento popis jsem u žádného diagramu neuvedl, jelikož by zabíral příliš mnoho stran a kopíroval by text vypsaný v kapitole s požadavky.

4.1.1 Operátor

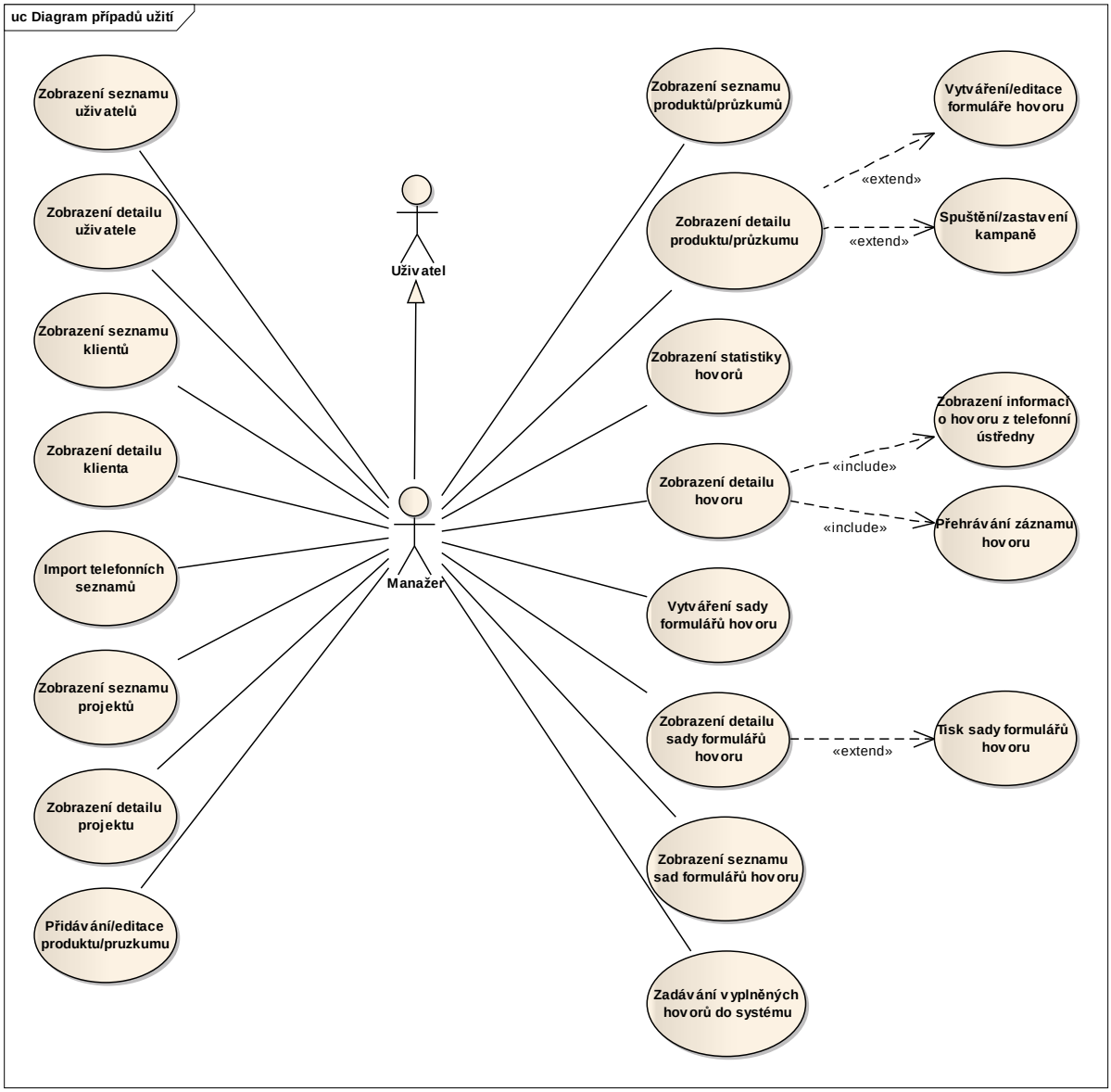
V následujícím diagramu je zobrazen případ užití pro operátory. Operátorem je v tomto diagramu myšlen operátor pracující na PC. Operátor bez PC spadá pod aktéra *Uživatel*.



Obrázek 4.1 Případ užití – operátor

4.1.2 Manažer

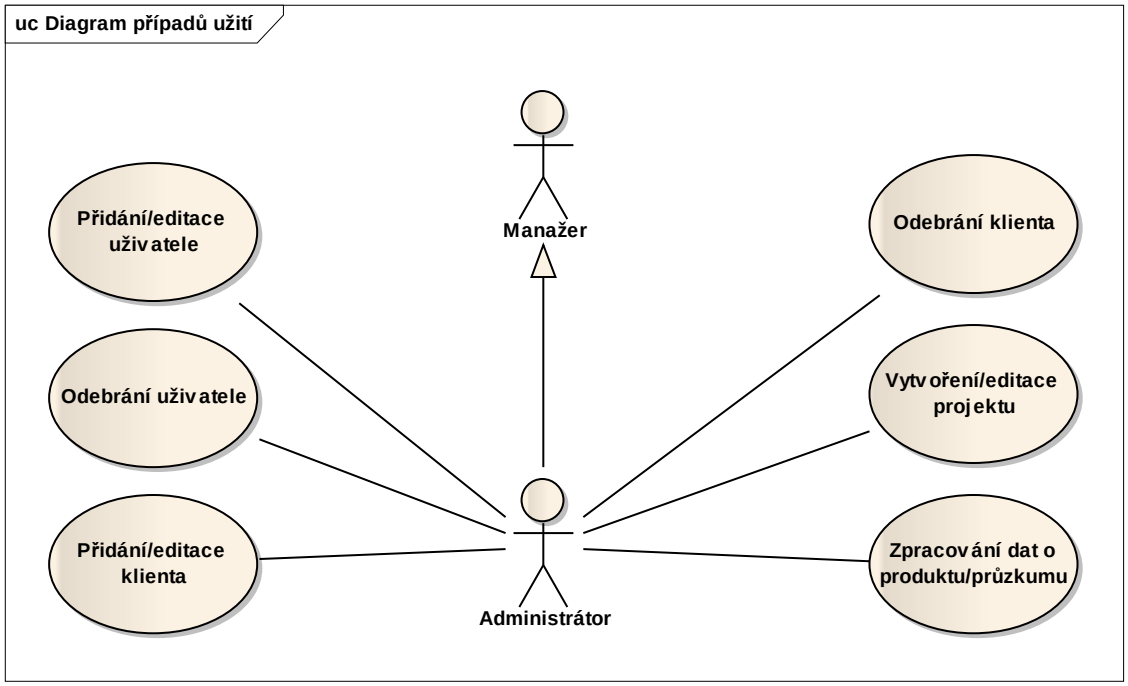
Na dalším diagramu je zobrazen případ užití uživatele s rolí manažer. Pro přehlednost zde již nejsou uvedeny případy užití pro aktéra *Uživatel*, které jsou stejné jako na předchozím diagramu.



Obrázek 4.2 Případ užití – manažer

4.1.3 Administrátor

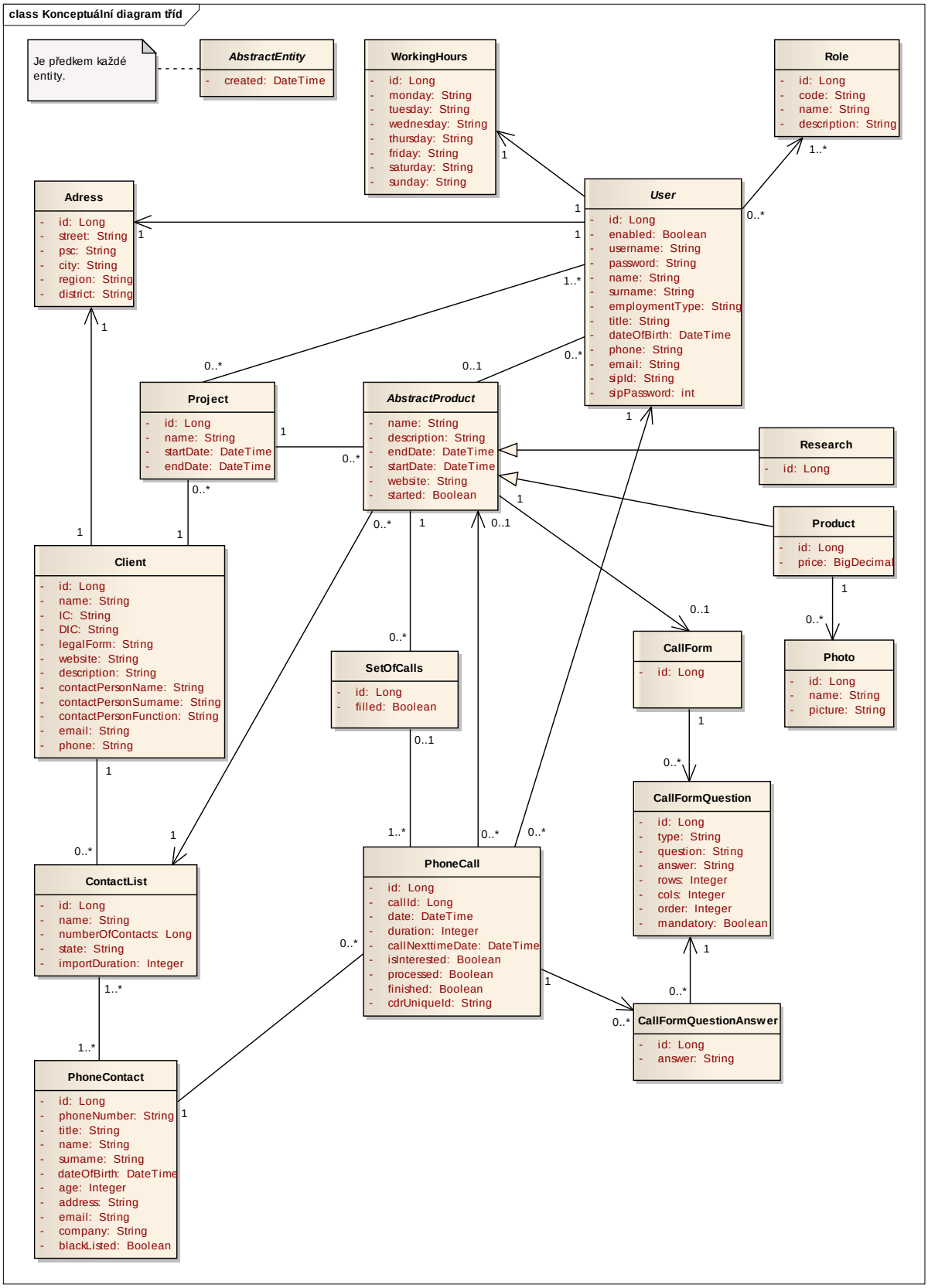
Na posledním diagramu je zobrazen případ užití uživatele s rolí administrátor. Pro přehlednost zde opět chybí případy užití pro aktéra *Uživatel*.



Obrázek 4.3 Případ užití – administrátor

4.2 Konceptuální diagram tříd

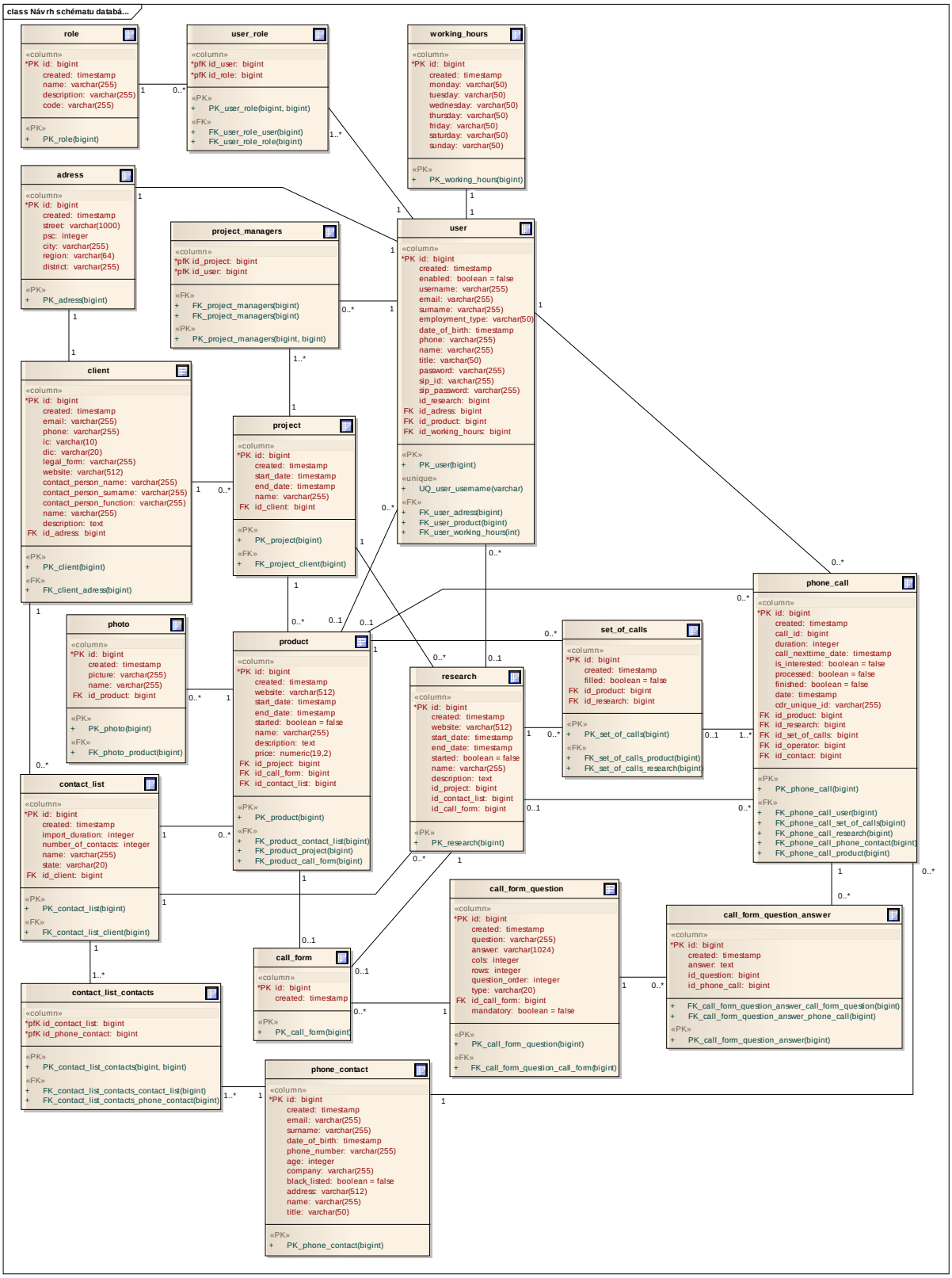
Konceptuální diagram tříd je základním diagramem UML pro modelování logického pohledu na statickou strukturu systému. Jelikož jsem si pro implementaci jádra aplikace vybral programovací jazyk Java Enterprise Edition, reprezentuje tento diagram perzistentní třídy v systému a jejich atributy s datovým typem jazyka Java. V rámci obecných programátorských zvyklostí budou všechny třídy a metody pojmenovány anglickými názvy.



Obrázek 4.4 Konceptuální diagram tříd

4.3 Návrh schématu databáze

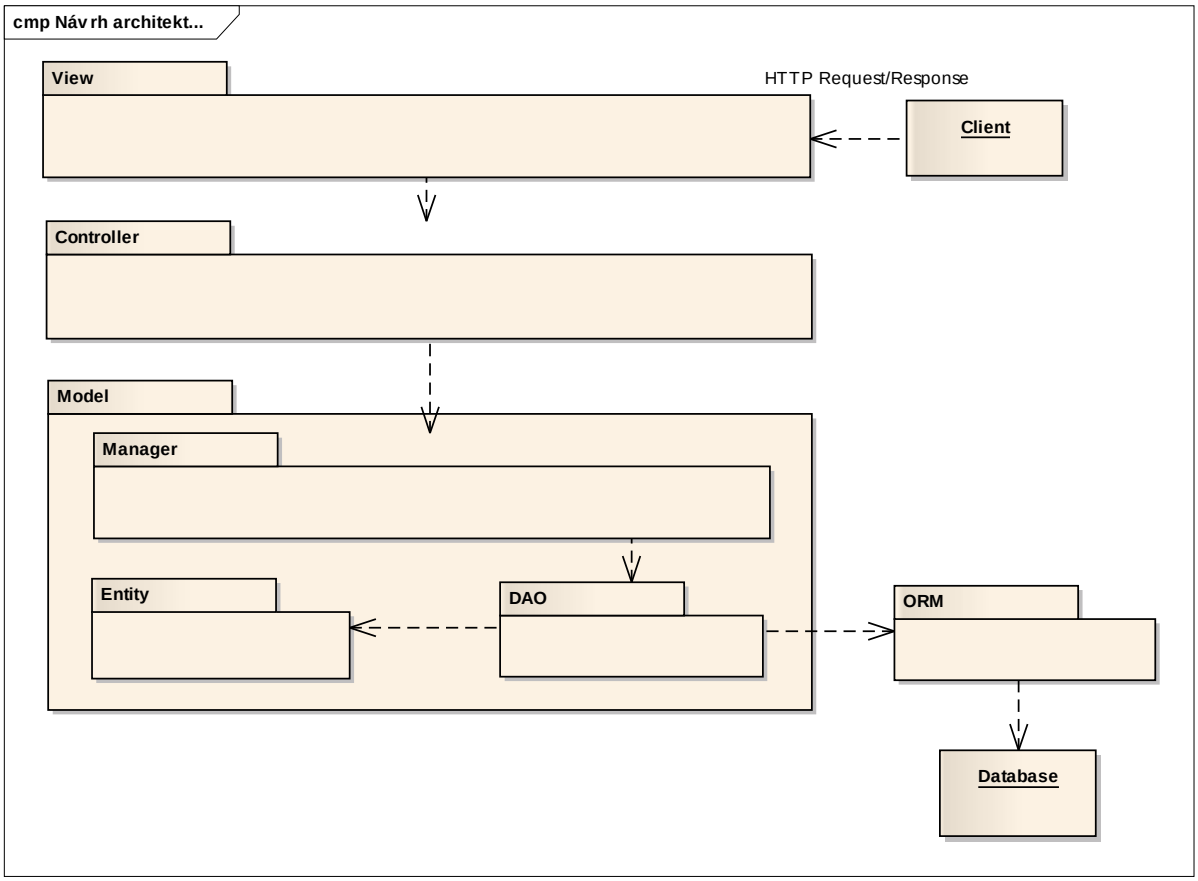
Pro uložení dat bude systém využívat klasickou relační databázi. Následující diagram schématu databáze ukazuje, jak budou jednotlivé perzistentní třídy uloženy na databázové úrovni.



Obrázek 4.5 Návrh schématu databáze

4.4 Návrh architektury

Architektura systému byla navržena tak, aby splňovala nejen všechny požadavky kladené na systém, ale aby byla zároveň moderní, robustní a škálovatelná. Jak jsem již zmínil dříve, hlavním programovacím jazykem, na kterém bude aplikace postavena, je Java Enterprise Edition. Uživatel bude s aplikací komunikovat prostřednictvím zasílání HTTP požadavků skrze webový prohlížeč s vrstvou View, která má na starosti zobrazování výsledných HTML stránek v prohlížeči. Tato vrstva je reprezentována šablonovací technologií JSP. Vrstva Controller zpracovává vstupy od uživatele, volá metody servisní vrstvy a volí šablonu (pohled) pro zobrazení dat. Vrstva Model tvoří jádro aplikace a skládá se ze tří částí – manažerů, entit a DAO vrstvy. Manažeři reprezentují tzv. servisní vrstvu, která obsahuje aplikační logiku a definuje transakční hranice, přičemž není nijak závislá na perzistentní vrstvě. Entity udržují datový model navržený v konceptuálním diagramu tříd v kapitole 4.2 a vrstva DAO neboli Data Access Objects spravuje spojení s databází a vykonává veškeré perzistentní operace. ORM je zkratkou pro Object Relational Mapping, což je vrstva zajišťující převod objektového datového modelu do relačního databázového modelu. Poslední vrstvou je tedy logicky relační databáze, jejíž schéma je navrženo v kapitole 4.3. Na obrázku 4.6 je zobrazen návrh architektury. Komunikace mezi jednotlivými vrstvami je jasně vymezena a bude takto striktně dodržována. Vrstva Controller tak např. nebude nikdy přistupovat přímo do vrstvy DAO, ale pouze do vrstvy Manager, která teprve potom přistoupí do vrstvy DAO.



Obrázek 4.6 Návrh architektury

5 Implementace systému

Tato kapitola popisuje konkrétní technologie a nástroje vybrané pro tvorbu aplikace. Jednotlivé nástroje nebudou rozebrány do příliš velkých detailů. Podrobné informace o nich lze získat ve zmiňovaných publikacích a internetových stránkách. Některé zajímavé detaily však budou podrobněji popsány v rámci popisu samotné implementace systému, která bude následovat v další části této kapitoly. Pro lepší pochopení důležitých faktů jsou některé podkapitoly doplněny obrázky. Kostra těchto obrázků je uvedená v anglickém jazyce kvůli zavedeným standardům a notacím a někdy je také doplněna českým komentářem. Ukázky zdrojových kódů, názvy tříd a metod jsou uvedeny písmem typu psací stroj. Názvy souborů, cesty v adresářové struktuře nebo některá jiná pojmenování jsou uvedena kurzívou.

5.1 Technologie a nástroje

Všechny vybrané nástroje použité pro implementaci a provoz systému jsou volně dostupné a stažitelné z internetu. Většina z nich je dokonce distribuována jako open source, tedy software s otevřeným zdrojovým kódem. V následujících podkapitolách budou všechny vybrané technologie a nástroje stručně popsány včetně vysvětlení mého důvodu jejich výběru.

Java Enterprise Edition

Java EE (dále pouze JEE) je v dnešní době již tradiční platforma pro vývoj informačních a podnikových systémů. Vychází z programovacího jazyka Java, který vyvinula firma Sun Microsystems. V současné době již tato firma neexistuje, jelikož v roce 2010 proběhla její akvizice firmou Oracle. Díky své dlouholeté tradici a vývoje těží JEE z velkého množství placených či volně dostupných nástrojů. V roce 2009 byla uvedena specifikace současné aktuální verze JEE 6. I přes velké množství rychle se rozvíjejících moderních technologií pro tvorbu informačních systémů se Java EE spolu s technologií .NET od firmy Microsoft řadí stále k nejpoužívanějším.

Aplikační rámec Spring

Použití samotné Javy EE nepřináší takovou efektivitu jako použití nějakého aplikačního rámce, který je na JEE postavený. Zvolil jsem tedy aplikační rámec Spring [3], který je současným hitem v oblasti vývoje JEE aplikací. Jedná se o open source projekt, který byl založen v roce 2003 Rodem Johnsonem na základě kódu publikovaného v jeho knize o vývoji bez EJB [4]. Důkazem obrovské úspěšnosti a popularitě tohoto projektu je fakt, že za 8 let své existence se postupně rozšířil do největšího komunitního Java projektu SpringSource [5], který aktuálně obsahuje několik programovacích jazyků, moderních aplikačních rámců, webových kontejnerů, aplikačních serverů či nástrojů pro nasazování aplikací do výkonných cloud systémů.

Samotný aplikační rámec Spring je modulární neinvazivní odlehčený kontejner, mezi jehož základní vlastnosti patří:

- **Kompletní podpora všech vrstev** – od datové vrstvy přes webové služby až po prezentační vrstvu či zabezpečení aplikace.
- **Integrace velkého množství kvalitních nástrojů třetích stran**
- **Inversion of Control** – návrhový vzor odstraňující těsné programové vazby mezi jednotlivými objekty a vrstvami.

- **Aspektově orientované programování** – vyčlenění některých částí stále se opakujícího kódu do tzv. aspektů, které mohou být poté volány před nebo po vykonání metody. Jedná se o jednu z nejsilnějších vlastností, která se prolíná celým rámcem.
- **Flexibilní MVC webový rámec** – umožňuje efektivní spolupráci s ostatními vrstvami aplikace a snadnou implementaci webových formulářů a zobrazování dat uživateli.

V aplikaci je použita verze 3.0.2. Současnou aktuální verzí je 3.0.5.

Hibernate

Hibernate [6] je nástroj, který umožňuje mapování objektů do tabulek relační databáze. V aplikaci je použit ve verzi 3.5 a reprezentuje vrstvu ORM, která je nastíněna v návrhu architektury aplikace v kapitole 4.4. Je přímo podporován rámcem Spring a poskytuje jednotné API pro manipulaci s daty nad databází. V aplikaci se Hibernate používá v tzv. Full-cream architektuře, kdy zajišťuje jak spojení s databází, tak provádění transakcí a poskytuje tak maximální odstínění od databázového rozhraní Javy JDBC. Hibernate je mocný nástroj, který jsem si zvolil ze dvou důvodů. Prvním důvodem je umožnění velmi snadné práce s databází na úrovni Java objektů a druhým je možnost použití složitých optimalizačních technik pro zajištění maximálního výkonu.

PostgreSQL

PostgreSQL [7] je volně dostupný multiplatformní relační databázový server s více než patnáctiletou historií rovněž distribuovaný jako open source. Zvolil jsem ho kvůli jeho široké použitelnosti a konfigurovatelnosti, vynikající stabilitě a výkonu. Tyto vlastnosti spolu s přívětivým uživatelským rozhraním pgAdmin III ho řadí mezi nejlepší volně dostupné databázové servery. V současné době je nejaktuálnější verze 9.0, nicméně v této práci je použita verze 8.3.

jQuery

JQuery [8] je javascriptová knihovna, která velmi usnadňuje procházení HTML dokumentem, manipulaci s událostmi, práci s technologií AJAX a vytváření vizuálních efektů. V současné době se mezi vývojáři webových aplikací začíná stávat standardem, jelikož se jedná o velice efektivní, rychlý, rozšířitelný a kvalitně zdokumentovaný nástroj umožňující kompletně separovat dynamické chování od statické struktury HTML dokumentu. Pomocí této knihovny lze implementovat spoustu funkcionality, která by v klasickém javascriptu byla velmi složitá a vyžadovala by velké množství kódu.

Direct Web Remoting

Běžná webová Java EE aplikace je založena na komunikaci s uživatelem na bázi HTTP požadavek – odpověď. V současné době je však trendem využívat ve webových aplikacích AJAX pro asynchronní komunikaci bez nutnosti znovu načítat danou stránku. AJAX je zkratkou pro asynchronní Javascript a XML. Jedná se o techniku asynchronní komunikace pomocí XML požadavků využívající Javascript pro odesílání a reflektování změn v prohlížeči. Direct Web Remoting [9] (dále jen DWR) je jedním z nástrojů, který umožňuje vzdálené volání Java metod pomocí AJAXu a také tzv. Reverse AJAX – vzdálené volání prohlížeče ze serveru. Je velmi jednoduchý na konfiguraci a používání a podporuje integraci do aplikačního rámce Spring. DWR není vhodné pro kompletní nahrazení komunikace přes

HTTP požadavky. Je ovšem ideální pro oživení webových stránek a implementaci některé jednoduché opakovaně využívané funkcionality jako např. našeptávání ve fulltextovém vyhledávání.

Asterisk

Součástí zadání diplomové práce je napojení vyvíjeného systému na telefonní ústřednu Asterisk. Jedná se o open source softwarovou implementaci telefonní ústředny, kterou vytvořil v roce 1999 Mark Spencer ze společnosti Digium, Inc., když nechtěl utrácet velké peníze za drahé hardwarové ústředny. V průběhu posledních deseti let se díky velmi početné komunitě stal Asterisk komplexním řešení poskytujícím VoIP ústřednu, pobočkovou ústřednu, voicemail, interaktivního hlasového průvodce, konferenční server a mnoho dalších služeb. Kompletní přehled možností, které Asterisk nabízí, lze vidět v představovacím videu na oficiálních webových stránkách projektu [10]. Díky nabízeným možnostem, vysoké modulárnosti, konfigurovatelnosti a zejména volné licenci je Asterisk v současné době jedno z nejlepších, nejflexibilnějších a nejrozšiřitelnějších řešení v oblasti integrovaných telekomunikačních systémů. Je schopen fungovat i na velmi slabém hardwaru a jeho využití je od jednoduchých domácích ústředen až po profesionální rozsáhlé systémy.

Apache Tomcat

Apache Tomcat je open source aplikační server implementující Java Servlet a JavaServer Pages technologie. Je vyvíjený společností Apache Software Foundation, která také stojí za vznikem populárního HTTP serveru Apache. JavaServer Pages (JSP) je technologie vytváření HTML kódu s dynamickým obsahem tvořeným Java kódem. Servlet je program napsaný v jazyce Java, který zpracovává požadavky a vytváří na ně odpovědi. Servlet překládá JSP stránky a výsledkem je HTML kód. V této práci tedy Tomcat tvoří úlohu serveru, na kterém je aplikace spuštěna. Použil jsem verzi 6.0, která implementuje specifikaci Servletu verze 2.5 a JSP verze 2.1. V současné době je již na webových stránkách projektu [11] stažitelná novější verze 7.0.

Apache Maven

Maven [12] je další nástroj od společnosti Apache Software Foundation, který představuje pokročilejší náhradu nástroje Ant [13]. Slouží ke kompilaci, distribuci, dokumentaci a celkové správě projektů v Javě. Zároveň ale také umožňuje použít klasické úkoly Antu. V aplikaci jsem použil verzi 2.1.0 pro kompilaci, generování databáze, sestavování projektu, správu potřebných knihoven třetích stran a generování JavaDoc dokumentace. V březnu 2011 byla vydána nejnovější verze 3.0.3, která však spíše místo nových funkcí přináší vyladění těch stávajících a opravu chyb [14].

Eclipse IDE

V závěru výčtu použitých technologií musím také zmínit vývojové prostředí Eclipse, ve kterém jsem celou aplikaci vytvářel. Eclipse je open source projekt od společnosti Eclipse Foundation stažitelný na webových stránkách [15]. Jedná se o vynikající vývojové prostředí určené zejména pro programování v jazyce Java EE, ale dnes již také pro C++ nebo PHP. Mezi jeho hlavní přednosti patří propracovaná kontrola syntaxe v průběhu psaní zdrojových kódů a podpora téměř neomezeného množství zásuvných modulů. Díky těmto modulům nemusí programátor externě používat ostatní nástroje, ale může si je integrovat přímo do vývojového prostředí a všechny operace provádět v něm. Na druhou stranu jsou tyto moduly také trochu nevýhodou v tom, že je každý vytvořen jinými programátory a mohou tak vznikat některé problémy s kompatibilitou. Mezi mé oblíbené moduly

patří Maven 2 plugin, Subversive, který integruje populární systém Subversion pro správu a verzování zdrojových kódů. Dále také Spring IDE, který zjednodušuje správu beanů rámce Spring a umožňuje vytvářet grafy s jejich závislostmi, což může někdy pomoci při orientaci ve velkých projektech.

5.2 Konfigurace aplikace

Základní konfigurace prostředí je definována v souboru *config.properties*, který se nachází ve složce *src/main/config*. Jsou zde uvedeny zejména proměnné pro připojení k databázi aplikace, telefonní ústředny a cesta ke složce pro ukládání souborů. Je to soubor, který centralizuje veškerá nastavení týkající se komunikace systému s jeho okolím. Pro správný chod systému tedy stačí nastavit pouze tyto proměnné. Ukázka tohoto souboru se nachází v příloze 1. Ostatní konfigurační soubory slouží k nastavení jednotlivých komponent systému a jejich vzájemné spolupráci.

5.2.1 Aplikační kontext

Jádrem celého systému je aplikační rámec Spring, který spolu propojuje velkou část použitých technologií. Většina konfigurace aplikace je tak realizována v rámci konfiguračních souborů rámce Spring. Tyto soubory se nacházejí se složce *src/main/webapp/WEB-INF/spring/* a jsou to konfigurační soubory aplikačního kontextu ve formátu XML. Aplikační kontext `org.springframework.context.ApplicationContext` je rozšířením továrny tříd `org.springframework.beans.factory.BeanFactory`, která zajišťuje vytvoření a provázání jednotlivých Java Bean objektů mezi sebou pomocí návrhového vzoru dependency injection [16]. Aplikační kontext navíc podporuje práci s externím zdrojem dat, publikování událostí, internacionalizaci v podobě zdroje zpráv a dědičnost z ostatních kontextů. Tato dědičnost umožňuje vytvořit více konfiguračních souborů pro různé části aplikace a v každém z nich používat odkazy na objekty definované v jiném souboru. Aplikační kontext je možné definovat jak programově, tak deklarativně. Využil jsem druhou možnost a to konkrétně ve formátu XML. V následujících podkapitolách budou zmíněny všechny konfigurační soubory aplikačního kontextu.

applicationContext-hibernate.xml

Tento soubor obsahuje veškerou konfiguraci datové vrstvy, která bude blíže popsána v kapitole 5.3.2.

applicationContext-business.xml

V tomto souboru se nachází základní nastavení aplikačního rámce Spring, který tvoří kostru celé aplikace. Ve starších verzích bylo zvykem všechny Java Bean objekty definovat ve formátu XML, což v případě větších aplikací vedlo k obsáhlým XML souborům. Od verze 2.5 je však možné veškerou konfiguraci realizovat pomocí Java anotací přímo ve zdrojových souborech daných Java Bean objektů. Tato metoda má jistě nesporné výhody v možnosti vytvoření konfigurace přímo při programování daného objektu bez nutnosti definice v XML souborech. Nevýhodu však spatřuji při provádění změn v konfiguraci, kdy nemusí být snadné najít zdrojový soubor obsahující danou konfiguraci, obzvlášť pokud nejsou zdrojové soubory pojmenovávány podle správných konvencí.

```
<context:component-scan base-package="telesystem.managers">
    <context:include-filter  type="annotation"
    expression="org.springframework.stereotype.Service"/>
</context:component-scan>
```

Uvedením této definice říkáme Springu, aby při spuštění aplikace prohledal balík `telesystem.managers` a všechny nalezené třídy s anotací `Service` instancioval jako Java Bean objekt. Anotace `Service` určuje, že se bude jednat o Java Bean objekt servisní vrstvy. Dalšími možnostmi jsou `Repository` (datová vrstva), `Controller` (prezentační vrstva rámce Spring MVC) a `Component` (obecný Java Bean objekt). Dále je v tomto konfiguračním souboru definován bean třídy `org.springframework.beans.factory.config.PropertyPlaceholderConfigurer`, který zajišťuje načtení výše zmíněného souboru *config.properties* a možnost využívání jeho proměnných jak ve zdrojových kódech, tak v XML konfiguračních souborech aplikačního kontextu. Dalším důležitým nastavením je zdroj zpráv, který se používá pro zajištění internacionalizace a umožňuje tak podle aktuálně zvoleného jazyka používat soubor s překlady pro tento jazyk. Soubor s překlady je ve formátu Java Properties, kde klíč je kód internacionalizační hlášky, která je použita v aplikaci a hodnota je její překlad. Systém telemarketingové společnosti sice nebyl vyvíjen pro vícejazyčnost, přesto jsem některé texty ukládal tímto způsobem. Jedná se o všechny hlášky, které jsou získávány již v Java kódu a musely byt tedy jinak být umístěny ve zdrojových souborech, což v případě hlášek v Českém jazyce není vhodný způsob. Pro zdroj zpráv jsem použil `org.springframework.context.support.ReloadableResourceBundleMessageSource` třídu, která umožňuje definovat interval pro znovu načítání souborů se zprávami a jejich kódování.

applicationContext-integration.xml

V tomto dalším konfiguračním souboru je nastavení integrace rámce Spring s dalšími technologiemi. Jedná se o značkovací knihovnu Freemarker [17] a knihovnu Flying Saucer [18], které se používají pro tisk do formátu PDF a také o knihovnu Direct Web Remoting. Všechny tyto technologie budou blíže popsány v následujících kapitolách.

applicationContext-security.xml

V souboru *applicationContext-security.xml* je veškeré nastavení zabezpečení aplikace pomocí bezpečnostního rámce Spring Security. Je zde zejména nastaven proces autentizace a autorizace a přístup uživatelských rolí na jednotlivé stránky.

webApplicationContext-xml

Posledním konfiguračním souborem aplikačního kontextu je *webApplicationContext-xml*, který, jak již z názvu vypovídá, slouží k nastavení prezentační vrstvy aplikace rámce Spring MVC.

5.2.2 Konfigurace webové aplikace

Hlavním konfiguračním souborem každé webové aplikace postavené na Java EE je soubor *web.xml*, který je uložený v adresáři *src/main/webapp/WEB-INF/*. Jedná se o první soubor, který se načítá při spuštění aplikace. Zde se nachází definice dvou servletů, filtrů webových požadavků, načítání výše popsaných konfiguračních souborů aplikačního kontextu a také časový interval vypršení uživatelského sezení. Servlet je vstupní bod pro webový požadavek. Prvním definovaným servletem je instance třídy `org.springframework.web.servlet.DispatcherServlet`, která zpracovává všechny klasické příchozí požadavky na aplikaci a dále je předává kontrolerům, které se postarají o jejich vyřízení. Druhým ze dvou definovaných servletů je instance třídy `org.directwebremoting.spring.DwrSpringServlet`, která zajišťuje zpracování všech požadavků s předponou */dwr/*. Jedná o požadavky na asynchronní vzdálené volání metod Javy. Dále

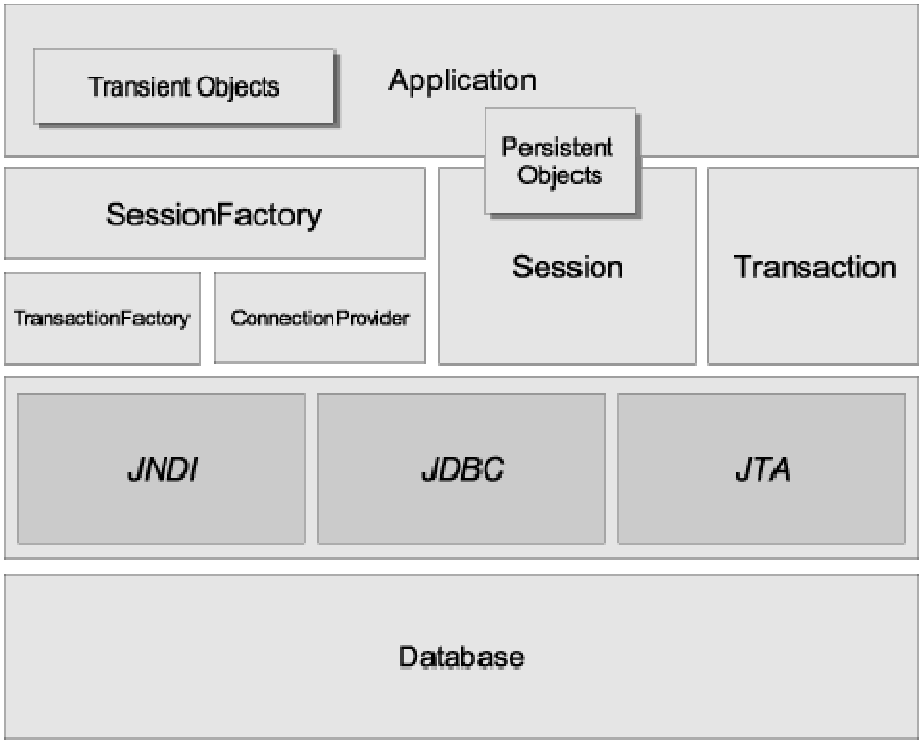
jsou zde ještě definovány přípony souborů, jejichž požadavky nemají být zpracovány žádným servetem. Jedná se hlavně o obrázky, kaskádové styly a soubory Javascriptu.

5.3 Datová vrstva

Datová vrstva je celá implementovaná pomocí rámce Hibernate, který je integrovaný do aplikačního rámce Spring. Reprezentují ji hlavně perzistentní a DAO objekty, ale prolíná se také se servisní vrstvou, kde jsou vytvářeny a uzavírány transakce.

5.3.1 Architektura Hibernate

Na obrázku 5.1 je znázorněna full-cream architektura Hibernate. V následujících podkapitolách vysvětlím tuto architekturu a jednotlivé části Hibernate v kontextu vyvíjené aplikace.



Obrázek 5.1 Full-cream architektura Hibernate

Pokud bychom obrázek rozdělili horizontálně do čtyř vrstev odspodu, tak spodní vrstva představuje relační databázi PostgreSQL. Další vrstva představuje standardní API poskytované Javou. V případě této aplikace je použito JDBC. Další vrstva představuje samotný Hibernate, jehož jádro se skládá z pěti nejdůležitějších rozhraní (*Session*, *Session Factory*, *Transaction* a *Configuration* a *Query*, které nejsou na obrázku znázorněny). Poslední horní vrstva představuje samotnou aplikaci. V tomto případě je to aplikace rámce Spring.

Rozhraní Session

Objekty rozhraní *Session* používá vlastní aplikace. Jsou to odlehčené objekty, které je levné vytvořit a zrušit. Vytvářejí se a zanikají velmi často, minimálně jednou za každý požadavek. Zapouzdřují v sobě

JDBC spojení a převod mezi objektovou a relační reprezentací dat. Zajišťují také vytváření transakcí a udržují vyrovnávací paměť první úrovně.

Rozhraní Session Factory

Objekty rozhraní *Session Factory* je naopak velmi nákladné vytvořit, a proto se vytvářejí pouze při startu aplikace. Jak už název napovídá, slouží jako továrna pro objekty rozhraní *Session*. Udržují v sobě také mapování perzistentních objektů na tabulky relační databáze a vyrovnávací paměť druhé úrovně. Zpravidla by pro každou databázi měla existovat vlastní instance *Session Factory*. Stejně tak tomu je i v této aplikaci a používají se zde celkem dvě instance. První instance je pro databázi PostgreSQL, která slouží jako hlavní databáze systému. Druhá instance je pro databázi MySQL, která je součástí telefonní ústředny a ukládá technické informace o provedených telefonních hovorech.

Rozhraní Transaction

Toto rozhraní je volitelnou součástí Hibernate a je možné implementovat vlastní vytváření transakcí. Doporučuje se však použít toto rozhraní, jelikož odstiňuje od vlastní implementace transakcí, která může být typu JDBC transakce, JTA transakce nebo dokonce Common Object Request Broker Architecture (CORBA) transakce. Toto rozhraní jsem zvolil pro použití s JDBC transakcemi.

Rozhraní Configuration

Rozhraní *Configuration* slouží ke správnému nastavení Hibernate a vytváření objektů *Session Factory*. V porovnání s ostatními rozhraními tvoří minimální část celého rámce, ale hraje velmi důležitou roli, protože nastavitelných položek je zde velké množství a dokážou velmi ovlivnit výkonnost a stabilitu celé datové vrstvy.

Rozhraní Query

Toto rozhraní poskytuje možnost vytváření databázových dotazů. V podstatě tu jsou tři možnost, jak vytvářet databázové dotazy. První možností je použití klasického nativního SQL jazyka. Druhou možností je použití Hibernate Query Language (HQL), což je dotazovací jazyk vycházející z klasického SQL s přidáním možnosti přímo pracovat s perzistentními objekty. Poslední možností je rozhraní Criteria, které je funkčností stejné jako HQL, ale liší ve způsobu vytváření dotazů, kdy se místo samotného dotazu postupně vytváří objekt nesoucí podobu dotazu ve svých atributech.

Hibernate tedy pracuje s aplikací pomocí objektů *Session*, které spravují perzistentní objekty. Perzistentní objekty jsou objekty, které jsou mapované na tabulky relační databáze a tvoří datový model aplikace. Tyto objekty se v rámci jedné session mohou nacházet ve třech různých stavech.

Přechodný stav

V tomto stavu se nachází objekty, které ještě nikdy nebyly asociovány s žádnou session a nemají tedy ani perzistentní identitu (mají prázdnou hodnotu primárního klíče). Jedná se o tzv. „transient“ objekty, které byly právě vytvořeny a čekají na první uložení do databáze.

Perzistentní stav

Jedná se o stav, ve kterém jsou všechny objekty asociované s danou session mající neprázdnou hodnotu primárního klíče a odpovídající řádek v tabulce databáze. U těchto objektů je zaručeno, že jejich objektová reprezentace v Javě je ekvivalentní s tou databázovou neboli, že obsahuje aktuální hodnoty z databáze. V tomto stavu se nachází většina objektů, se kterými Hibernate pracuje.

Oddělený stav

V tomto stavu se nachází objekty, které byly někdy asociované s nějakou session, ale ta už není k dispozici. Jedná se také o objekty, které mají neprázdnou hodnotu primárního klíče a pravděpodobně i odpovídající řádek v tabulce databáze. Není u nich však zaručeno, že obsahují aktuální hodnoty. Do tohoto stavu se objekty v aplikaci mohou dostávat poměrně často a je potřeba zajistit, aby se s nimi dále nepracovalo a aby byly znovu asociovány s danou session a obsahovaly tak aktuální hodnoty. Více o rámci Hibernate se lze dozvědět v knize [19], ze které jsem čerpal.

5.3.2 Konfigurace Hibernate

Jak již bylo zmíněno v kapitole 5.2.1, tak konfigurace datové vrstvy a tedy i Hibernate je uložena v souboru *applicationContext-hibernate.xml*. Je zde definován bean pro vytváření transakcí, nastavení rozhraní *Session Factory* a zdroje dat. Aplikace pracuje se dvěma databázemi. První z nich běží na PostgreSQL a slouží jako databáze samotného systému. Druhá z nich běží na databázovém stroji MySQL a je to databáze telefonní ústředny, kam se ukládají technické informace o telefonních hovorech. Každá z těchto databází využívá vlastní *Session Factory*, a proto jsou v konfiguraci uvedeny celkem dvě. Jelikož je v aplikačním kontextu načten konfigurační soubor prostředí *config.properties*, tak z něho lze čerpat proměnné i v tomto nastavení.

V následujících řádcích budou uvedeny některé důležité vlastnosti z nastavení *Session Factory*.

```
<prop key="hibernate.dialect">${hibernate.dialect}</prop>
<prop key="hibernate.jdbc.batch_size">20</prop>
<prop key="hibernate.hibernate.generate_statistics">true</prop>
<prop key="hibernate.cache.use_second_level_cache">true</prop>
<prop key="hibernate.cache.use_query_cache">true</prop>
```

Vlastnost `hibernace.dialect` určuje třídu, která bude zajišťovat optimální převádění HQL dotazů do SQL. Hibernate poskytuje několik těchto tříd, kde každá je určena pro jiný databázový stroj. Zde je právě použita proměnná ze souboru *config.properties*. `hibernate.jdbc.batch_size` určuje počet příkazů update nebo insert, které je možné provést v jedné dávce najednou. Doporučené nastavení z hlediska výkonnosti je okolo 20. Vlastnost `hibernate.generate_statistics` definuje, zda má Hibernate sbírat statistiky o všech provedených dotazech, které mohou sloužit k ladění výkonnosti. `hibernate.cache.use_second_level_cache` umožňuje zapnout vyrovnávací paměť druhé úrovně, která se používá zejména pro uchovávání perzistentních tříd. `hibernate.cache.use_query_cache` umožňuje zapnout vyrovnávací paměť pro výsledky databázových dotazů. Pro její korektní funkčnost je však ještě nutné vždy u daného dotazu explicitně uvést, že má tuto paměť použít. Více o vyrovnávací paměti v Hibernate bude uvedeno v kapitole 5.7.2.

V nastavení zdroje dat jsou definovány URL, přihlašovací jméno a heslo pro připojení k databázi a také databázový ovladač. Všechny tyto nastavení využívají proměnné z *config.properties* a tím je jednoduše docíleno databázové nezávislosti. Nastavením jiného ovladače, dialektu, URL a přihlašovacích údajů lze změnit databázový stroj bez nutnosti dalších zásahů do aplikace. Databáze systému používá udržování spojení – tzv. „connection pooling“ pomocí Java knihovny c3p0 [20].

```
<property name="minPoolSize" value="15" />
<property name="maxPoolSize" value="60" />
<property name="maxIdleTime" value="300" />
<property name="idleConnectionTestPeriod" value="150" />
```

`minPoolSize` určuje minimální počet spojení s databází, která budou vždy otevřena. Naopak `maxPoolSize` určuje maximální počet, který nebude nikdy překročen. Vlastnost `maxIdleTime` definuje, za jak dlouho (v sekundách) bude ukončeno spojení, které již není aktivní. `idleConnectionTestPeriod` určuje interval (v sekundách), ve kterém se bude kontrolovat nečinnost jednotlivých připojení. Jelikož vytvoření nového spojení je relativně pomalé, tak správná volba těchto hodnot v závislosti na charakteristice systému může vést k jeho výrazně lepší výkonnosti. V příloze 2 je uvedeno celé nastavení *Session Factory* a zdroje dat pro databázi systému. Nastavení pro databázi telefonní ústředny je téměř shodné. Její zdroj dat pouze nepoužívá udržování spojení, jelikož tato databáze není využívána tak často.

5.3.3 Mapování perzistentních objektů

V celé aplikaci se pracuje s perzistentními objekty pouze v podobě klasických Java objektů. Hibernate však musí zajišťovat správné ukládání těchto objektů do tabulek relační databáze. To, do které tabulky a do kterých sloupců jaká data uloží, určuje tzv. mapování perzistentních objektů. Toto mapování musí definovat sám vývojář buď pomocí XML souborů, nebo Java anotacemi. První ze zmíněných možností je zastaralá a dnes se již příliš nepoužívá. Zvolil jsem tedy modernější způsob pomocí Java anotací.

V následující ukázce budou popsány některé anotace sloužící k mapování třídy reprezentující projekt.

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name = "project", schema = "public")
@TypeDefs( { @TypeDef(name = "jodaDateTime",
    typeClass = PersistentDateTime.class) })
@SequenceGenerator(name = "project_id_sequence",
    sequenceName = "project_id_sequence", initialValue = 20,
    allocationSize = 1)
public class Project extends AbstractEntity {
    …
    private Long id;
    private Set<User> managers;
    …
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
        generator = "project_id_sequence")
    public Long getId()
    …
    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "project_managers", joinColumns = @JoinColumn(
        name = "id_project"), inverseJoinColumns = @JoinColumn(
        name = "id_user"))
    public Set<User> getManagers()
```

První typ anotací popisujících vlastnosti databázové tabulky se uvádí nad definicí třídy. `@Entity` určuje, že tato třída je perzistentní a má tedy svoji databázovou reprezentaci. Anotace `@Inheritance` definuje, jakým způsobem budou mapovány atributy vyplývající z dědičnosti třídy `Project` z třídy `AbstractEntity`. Strategie *JOINED* zajišťuje, že tyto atributy budou v tabulce projektu. V anotaci `@Table` je uvedeno schéma a název tabulky, ve které budou hodnoty objektu uloženy. `@TypeDefs` umožňuje definovat třídy pro konverzi nestandardních datových typů.

Hibernate nativně podporuje konverzi běžných datových typů jazyka Java. V celé aplikaci je však použita knihovna Joda Time [21] pro pohodlnější práci s datem. Převod data z této knihovny do databázové podoby Hibernate nativně nepodporuje. Díky definici `@TypeDef`, která zaregistruje konverzní třídu z knihovny Joda Time, je to však umožněno. `@SequenceGenerator` popisuje generátor sekvencí s názvem `project_id_sequence` s iniciální hodnotou 20 a získáváním nového čísla sekvence z databáze po každém uloženém záznamu.

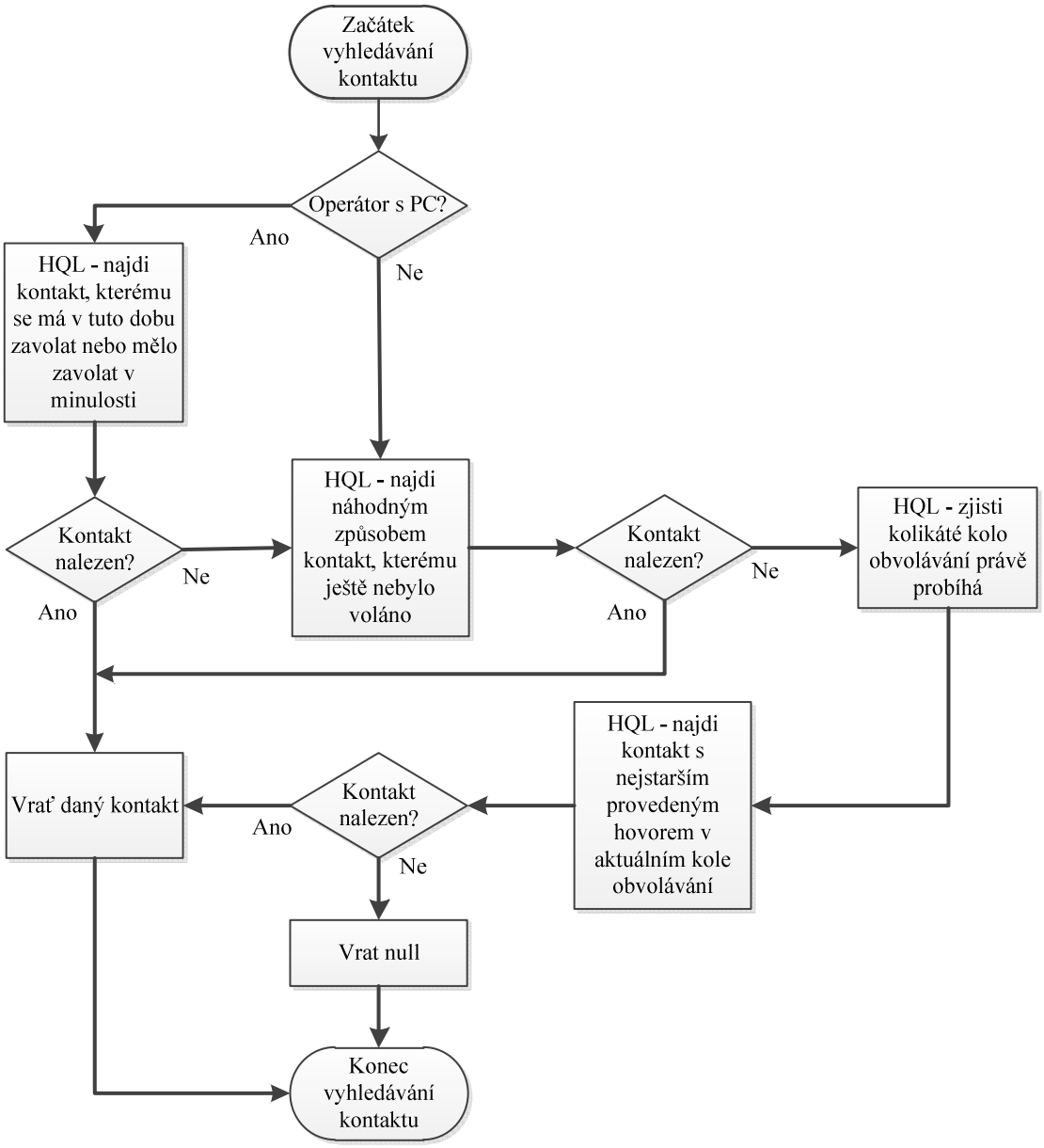
Druhý typ anotací popisuje daný atribut, a proto se také definuje buď přímo u tohoto atributu nebo v jeho get metodě jako v tomto případě. Anotace `@Column` značí, že atribut bude v dané tabulce uložen ve sloupci s názvem v proměnné `name`. `@Id` označuje primární klíč tabulky a `@GeneratedValue` způsob, jakým bude generován pro nové záznamy. V tomto případě se použije výše definovaná sekvence. Poslední ukázkou je mapování vztahu projekt-manažer s kardinalitou M:N, kde `@ManyToMany` určuje právě tuto kardinalitu. Vlastnost `FetchType.LAZY` zaručuje načítání jednotlivých položek této kolekce až ve chvíli, kdy s ní bude poprvé manipulováno. To většinou produkuje lepší výkonnost, ale musí se ošetřovat případy, kdy již session udržující tuto kolekci byla uzavřena. Opakem je okamžité načítání celé kolekce z databáze při načtení tohoto objektu. `@JoinTable` definuje vazební tabulku vztahu M:N.

Výše popsaným způsobem je namapován celý datový model zobrazený v kapitole 4.2. Hibernate také poskytuje podpůrný projekt s názvem Hibernate Tools [22], který kromě usnadněné práce s tímto nástrojem ve vývojovém prostředí Eclipse, také umožňuje generovat SQL příkazy pro vytvoření schématu celé databáze na základě mapování perzistentních objektů.

5.3.4 Vybírání telefonního kontaktu

V této kapitole bude popsána nejsložitější funkcionalita datové vrstvy. Jedná se o vybírání telefonního kontaktu z telefonního seznamu pro nový hovor operátora, které je popsáno v požadavku č.11 v kapitole 3.3. Celou tuto funkcionalitu provádí metoda `getPhoneContactForProduct` třídy `ContactListDao.java`. Jedná se v podstatě o několik HQL dotazů, které jsou volány podle logiky znázorněné na obrázku 5.1.

Rozhodnutí, zda se jedná o operátora s PC, závisí na tom, zda se tato metoda volá při vytvoření nového systémového hovoru pro operátora pracujícího na PC nebo při vytváření sady formulářů hovoru pro operátory bez PC. Kolo obvolávání reprezentuje počet provedených hovorů všem telefonním kontaktům v seznamu v rámci dané kampaně. Vybrání kontaktu s nejstarším provedeným hovorem v aktuálním kole tedy zajišťuje, aby se nikdy nevolalo některému kontaktu vícekrát po sobě v krátkém časovém intervalu. Případ, kdy není žádný kontakt nalezen a je vrácen `null`, by teoreticky neměl nikdy nastat, jelikož počet kol obvolávání není nijak omezený.



Obrázek 5.1 Logika volání HQL dotazů pro vybrání telefonního kontaktu

5.4 Servisní vrstva

Metody servisní vrstvy jsou volány z prezentační vrstvy a vykonávají aplikační logiku nebo dále volají metody datové vrstvy. Třídy reprezentující servisní vrstvu se nacházejí v balíku *telesystem/managers* a jsou rozděleny podle perzistentních tříd, kterých se týká jejich funkcionalita. Je zde obsažen klasický Java kód a z pohledu implementace je zde zajímavé pouze vytváření transakcí.

V konfiguračním souboru *applicationContext-business.xml* jsem definoval transakčního manažera aplikačního rámce Spring pro Hibernate.

```
<bean id="transactionManagerBean"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
   <property name="sessionFactory" ref="sessionFactory"/>
</bean>
```

```
<tx:annotation-driven transaction-manager="transactionManagerBean"/>
```

Tento transakční manažer umožňuje deklarativní správu transakcí na úrovni servisní vrstvy. Transakční zpracování lze definovat v XML konfiguraci nebo pomocí anotací jak pro celou třídu, tak pro jednotlivé metody. Jelikož jedna metoda této vrstvy může volat jinou metodu této vrstvy, tak jsou podporovány i vnořené transakce, přičemž lze nastavit celkem 7 způsobů jejich propagace.

```
@Service("callFormManager")
@Transactional(propagation = Propagation.REQUIRED)
public class CallFormManagerImpl implements CallFormManager {

    @Autowired
    private CallFormDao callFormDao;
```

Výše uvedený kód ukazuje nastavení transakcí pro třídu manažera poskytujícího metody pro práci s formulářem hovoru. Anotace `@Service` definuje, že se jedná o Java bean servisní vrstvy s názvem `callFormManager`. Anotace `@Transactional` uvedená takto nad definicí třídu určuje, že pro všechny metody této třídy budou vytvářeny transakce s propagací typu *REQUIRED*. Použitím anotace `@Autowired` Spring automaticky propojí tuto třídu manažera s třídou datové vrstvy a umožní tak volat její metody.

V následující části je popsáno všech 7 možných způsobů nastavení propagace.

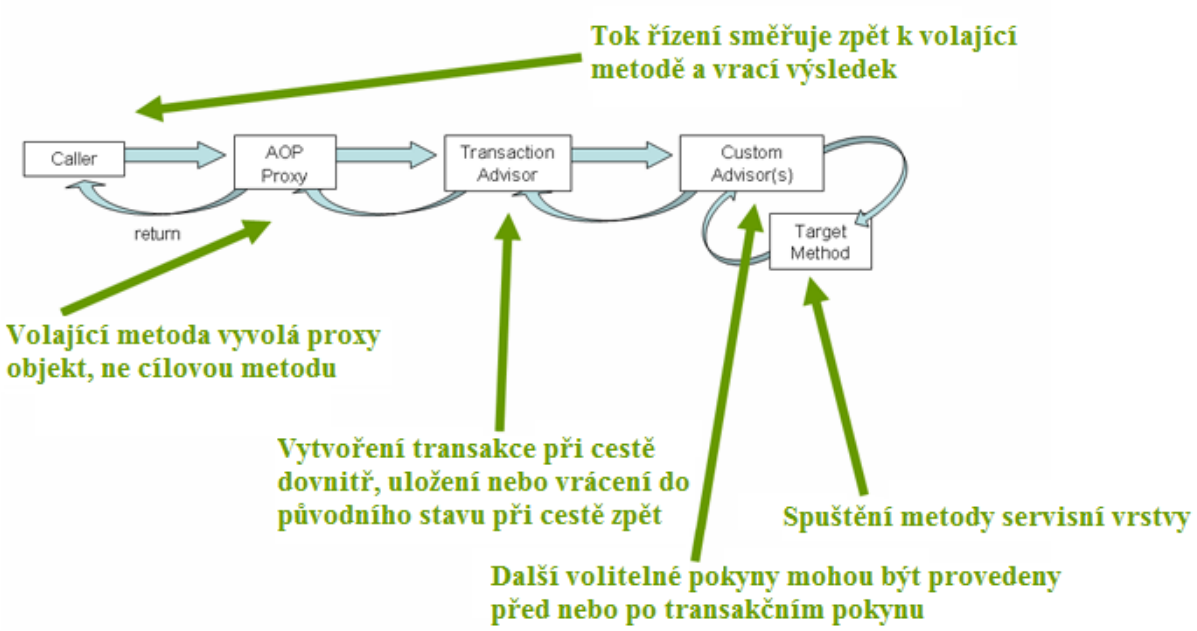
1) MANDATORY – pokračuje v aktuální transakci, pokud žádná neexistuje, tak vyhodí výjimku
2) NESTED – pokud existuje transakce, tak vytvoří vnořenou transakci, jinak vytvoří novou
3) NEVER – vykoná metodu bez transakce, pokud však již nějaká existuje, tak vyhodí výjimku
4) NOT_SUPPORTED – vykoná metodu bez transakce, pokud však již nějaká existuje, tak ji pozastaví
5) REQUIRED – pokračuje v aktuální transakci, pokud žádná neexistuje, tak vytvoří novou
6) REQUIRES_NEW – vždy vytvoří novou transakci, pokud již nějaká existuje, tak ji pozastaví
7) SUPPORTS – pokračuje v aktuální transakci, pokud žádná neexistuje, tak vykoná metodu bez transakce

Běžnou praxí bývá globální nastavení typu *REQUIRED* pro celou třídu a případné nastavení výjimek pro konkrétní metody.

Další důležitou nastavitelnou vlastností transakcí je jejich úroveň izolace, která určuje, jaké změny budou viditelné ostatním transakcím. V ideálním případě by měla být každá transakce izolovaná od ostatních a změny, které provede, by tedy měly být viditelné až po jejím ukončení. Tento přístup však vyžaduje sekvenční zpracování transakcí manipulujících se stejnými daty a to může mít negativní dopad na výkonnost systému. V některých aplikacích nemusí být vyžadována úplná izolace, a tak Spring nabízí celkem 5 možností nastavení izolace transakcí. S těmi možnostmi propagace a izolace lze pohodlně a optimálně vyladit nastavení transakčního zpracování. Implicitně ostatní transakce nevidí změny provedené v aktuální transakci, dokud není ukončena. Samotná vykonávaná transakce však může ve svém průběhu pracovat s daty, která jsou výsledkem jiných aktuálně ukončených transakcí. Je tedy možné, že v různých místech jejího průběhu bude pracovat s jinými daty. Nebudu zde již popisovat ostatní možnosti, které lze nalézt v knize [23], z níž jsem čerpal.

Deklarativní správa transakcí v rámci Spring je umožněna díky využití principu aspektově orientovaného programování [24] a vytváření proxy objektů. Na obrázku 5.2 je znázorněno, jakým

způsobem probíhá volání metody, která je pod touto správou. Obrázek jsem převzal z dokumentace rámce [25] a doplnil jej českým překladem popisu jednotlivých částí.



Obrázek 5.2 Volání metody pod transakční správou

Samozřejmě, že Spring také podporuje imperativní správu transakcí, kdy je možné, aby si vývojář přímo definoval jejich vytváření, ukládání a návraty do původního stavu.

5.5 Prezentační vrstva

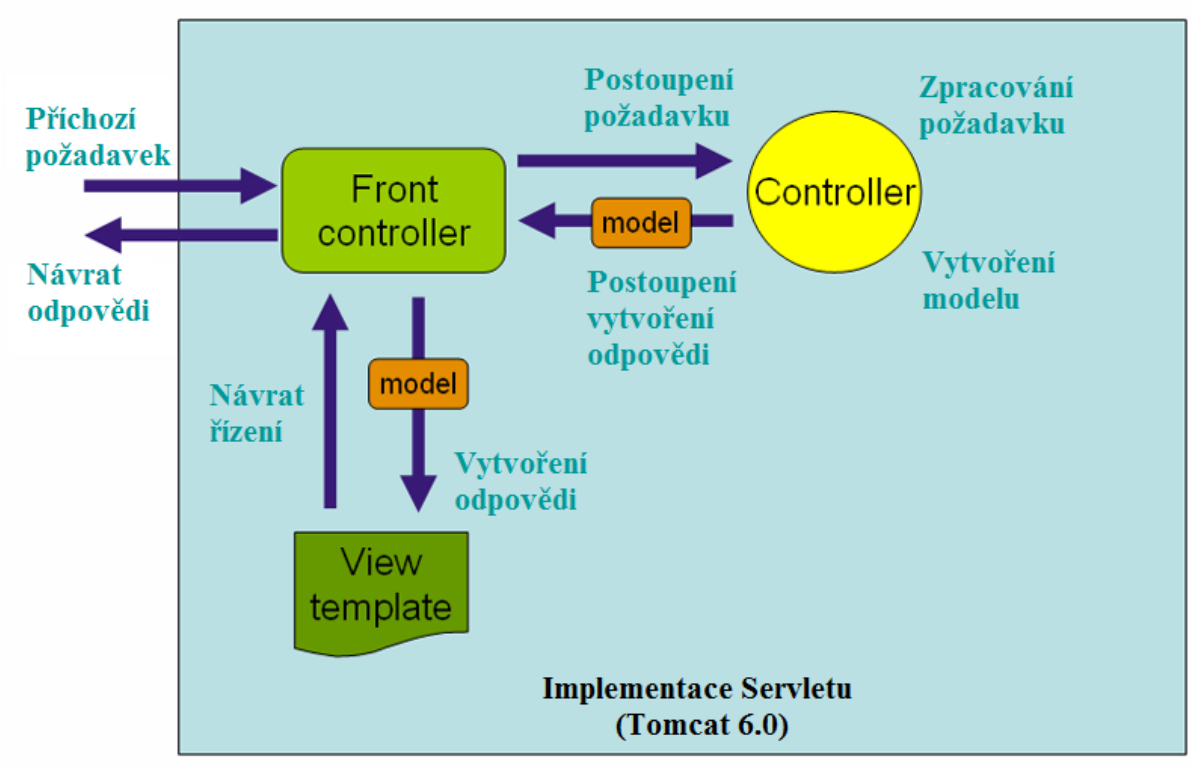
Prezentační vrstva je poslední vrstvou stojící mezi uživatelem a aplikací. Jak už název vypovídá, tak jejím úkolem je zpracovávat vstupy uživatele a prezentovat výsledky. Pro realizaci této vrstvy jsem použil rámec Spring Web MVC doplněný vzdáleným voláním metod Javy pomocí nástroje DWR a Javascript knihovny jQuery. Komunikačním prvkem této vrstvy je webový prohlížeč a výstupy jsou HTML stránky nebo v případě tisku formulářů hovoru také PDF dokument.

5.5.1 Spring Web MVC

Spring Web MVC je součástí aplikačního rámce Spring a slouží k implementaci prezentační vrstvy. Samotné jádro Springu poskytuje podporu pro další rámce třetích stran, ale Spring Web MVC má výhodu v plné integraci a možnosti využít všechny přednosti rámce Spring. Je postaven na návrhovém vzoru Model-View-Controller, který rozděluje tuto vrstvu na další 3 části:

- **Model** – je tvořen objekty nesoucími data, která budou zobrazena prostřednictvím pohledu
- **View** (pohled) – reprezentuje odpověď na požadavek, přijímá od kontroleru model a zobrazuje jej
- **Controller** (kontroler) – zpracovává požadavek a na jeho základě vytvoří model, který předá pohledu

Jak je vidět, tak hlavním řídícím prvkem, který jednotlivé části spojuje, je kontroler.



Obrázek 5.3 Zpracování požadavku v Spring Web MVC

Dalším návrhovým vzorem, na kterém je Spring Web MVC postaven, je tzv. Front Controller. Tento návrhový vzor definuje, jakým způsobem bude každý nový požadavek zpracován. V případě návrhového vzoru Front Controller existuje jeden hlavní kontroler, který přijímá všechny požadavky a rozhoduje, jaký další kontroler bude zavolán pro jejich zpracování. V rámci Spring Web MVC je tímto hlavním kontrolerem `DispatcherServlet`. Na obrázku 5.3 je zobrazen tok požadavku v návrhovém vzoru Front Controller. Obrázek jsem opět převzal z dokumentace [26] a doplnil jsem českým překladem jednotlivých částí. Ve vyvíjeném systému zpravidla existuje pro každou stránku vlastní kontroler, který ji zpracovává. Výjimku tvoří přihlašovací, úvodní stránka a stránka pro zobrazení nepovoleného přístupu. Tyto tři jsou zpracovávány jedním společným kontrolerem, jelikož nevyžadují žádnou složitou logiku.

Nedílnou součástí tohoto rámce je také JSP knihovna značek (spring tag, form tag), která usnadňuje získávání dat z formulářů, validaci chyb nebo zobrazování internacionalizačních hlášek. JSP značky mají podobu HTML značek, tedy počáteční a koncovou značku a atributy. Každá JSP značka je reprezentován Java třídou a při překládání servletem jsou zavolány její metody. Tímto způsobem lze jednoduše spouštět rozsáhlý Java kód uvnitř JSP stránky při zachování její minimální velikosti a tím pádem také přehlednosti.

Kontrolery můžeme rozdělit do dvou kategorií – klasické a formulářové. V následující části bude uveden popis a zjednodušená ukázka implementace klasického kontroleru pro zobrazení detailu uživatele systému a formulářového kontroleru sloužícího k vytvoření a editaci uživatele.

Klasický kontroler

Tento typ kontroleru je jednodušší na implementaci i konfiguraci a slouží pouze k zobrazování dat.

```
@Controller("userDetailController")
@RequestMapping("/uzivatel/{id}")
public class UserDetailController {
    // definice injektovaných beanů
    public UserDetailController() {
        view = "/users/user-detail";
    }

    @RequestMapping(method = RequestMethod.GET)
    public String handle(@PathVariable Long id, HttpServletRequest request,
    ModelMap map) {
        // zpracování požadavku a sestavení modelu
        return view;
    }
```

Anotace `@Controller` definuje, že tato třída bude kontrolerem s názvem `userDetailController` a bude tedy zařazena mezi kandidáty na zpracování požadavků postoupených z třídy `DispatcherServlet`. Anotace slouží k mapování požadavků zaslaných z určitého vzoru URL adresy na daný kontroler nebo přímo metodu kontroleru. V tomto případě říká, že na tento kontroler budou směřovat všechny požadavky z URL `/uzivatel/{id}`. `{id}` zde reprezentuje řetězec, který potom ve zpracovávající metodě bude přístupný pod proměnnou `id`. V konstruktoru třídy je definován pohled, který bude použit k vytvoření odpovědi. V konfiguračním souboru rámce Spring Web MVC *webapplicationContext.xml* je definovaný bean `jstlViewResolver`, který zajišťuje vyhodnocení JSP pohledů. Pokud není určeno jinak, tak se implicitně zadaný pohled vyhodnocuje jako JSP stránka, která je uložena v cestě nastavené u toho beanu a má příponu *jsp*. V této aplikaci jsou všechny JSP stránky uloženy ve složce */WEB-INF/jsp*. Znovupoužitá anotace nad metodou `handle` určuje, že tato metoda bude volána pro všechny požadavky odeslané HTTP metodou GET. Uvnitř metody `handle` pak proběhne samotné zpracování požadavku a sestavení modelu, který bude vložen do vráceného pohledu. Pokud by byl vyvolán požadavek na tento kontroler jinou HTTP metodou (např. POST), tak by `DispatcherServlet` vyhodil výjimku, že daná metoda není podporována.

Formulářový kontroler

Formulářové kontrolery slouží k získání dat od uživatele prostřednictvím webových formulářů. Typickým příkladem je kontroler pro vytvoření a editaci uživatele, kde jsou data vkládána pomocí textových, zaškrtávacích a výběrových polí.

```
@Controller("userCreateFormController")
@SessionAttributes("command")
public class UserCreateFormController extends AbstractFormController {
    // definice injektovaných beanů
    public UserCreateFormController() {
        setFormView("users/user-create");
        setSuccessView("redirect:/uzivatele/vytvorit/dokoncit/{0}");
    }

    @RequestMapping(value = {"/uzivatele/vytvorit",
    "/uzivatele/editovat/*"}, method = RequestMethod.GET)
    public String showForm(HttpServletRequest request, ModelMap map) {
        UserCreateForm form = new UserCreateForm();
        // zpracování požadavku a sestavení modelu
        map.addAttribute("command", form);
```

```
        return getFormView();
    }

    @RequestMapping(method = RequestMethod.POST)
    public String processSubmit(HttpServletRequest request, ModelMap map,
    @ModelAttribute("command") UserCreateForm form, BindingResult result,
    SessionStatus status) {
        // validace a uložení dat z formuláře
        return getSuccessView();
    }
```

Anotace `@SessionAttributes` definuje ukládání atributu modelu s názvem `command` do uživatelského sezení a tím pádem jeho dostupnost mezi jednotlivými požadavky. Tento atribut se nazývá „command object“ a reprezentuje formulář, do kterého se budou ukládat data. Metoda `showForm` je volána při zaslání požadavku HTTP metodou GET a tedy při zobrazení webové stránky pro vytvoření či editaci uživatele. V této metodě se inicializuje formulářový objekt a vkládá se do modelu. V případě vytváření uživatele se zde tedy vytvoří nový objekt `User`, v případě editace se získá instance editovaného objektu `User` z databáze a vloží se do formuláře. Návratovou hodnotou je řetězec s pohledem reprezentujícím JSP stránku obsahující pole pro vložení dat. Metoda `processSubmit` je zavolána po odeslání formuláře HTTP metodou POST. Definicí anotace `@ModelAttribute` je umožněno přistoupit k formulářovému objektu s vyplněnými daty od uživatele. V JSP stránce je použit JSP form tag rámce Spring Web MVC, který navazuje jednotlivá pole pro vložení dat na atributy formulářového objektu a zajišťuje tedy i jejich správný převod na datové typy jazyka Java a snadnější validaci. Ihned po vstupu do této metody je provedena validace uživatelem vložených dat a v případě nalezení chyb je vrácen stejný pohled jako v metodě `showForm` a uživateli je zobrazen seznam validačních chyb pomocí JSP form tagu. Po úspěšné validaci jsou data zpracována a poslána do metody servisní vrstvy, která zajistí jejich uložení voláním metod datové vrstvy. Návratovou hodnotou zde už ovšem nesmí být JSP stránka, ale přesměrování na některou URL adresu. Kdyby zde byla uvedena opět JSP stránka, tak by při jejím znovu zobrazení došlo ke znovu odeslání formuláře a to jistě není žádoucí. Uvedením prefixu `redirect:` Spring provede přesměrování odesláním odpovědi s HTTP statusem 302. Řetězec `{0}` v URL pro přesměrování reprezentuje proměnnou, za kterou je dosazen identifikátor nově vytvořeného, případně editovaného uživatele.

Stejně jako ostatní části rámce Spring i jednotlivé kontrolery lze nakonfigurovat také v XML souborech aplikačního kontextu. Tento způsob se používal ve starších verzích Springu a implementace samotných kontrolerů byla trochu odlišná. Bylo zde nutné, aby každý kontroler rozšiřoval některý základní z rámce Spring a používal pro zpracování požadavku jeho metody. Je to sice zastaralý způsob, který je v současné době považován za nemoderní, ale dle mého názoru byl lépe konfigurovatelný a lépe se dokázal vypořádat s některými nestandardními stavy. Předpokládám však, že v nadcházejících verzích budou tyto rozdíly úplně smazány a konfigurace pomocí anotací přinese již pouze samé výhody.

5.5.2 Vzdálené volání metod Javy

Jak jsem již zmínil dříve, tak komunikace uživatele se systémem neprobíhá pouze odesíláním HTTP požadavků a zobrazováním celých HTML stránek, ale také voláním vzdálených metod jazyka Java bez nutnosti znovu načítání celých stránek. Tohoto principu je docíleno použitím knihovny Direct Web Remoting a jQuery. Obě tyto technologie byly představeny již v kapitole 5.1 a nyní zde bude

uvedeno, jakým způsobem pracují a jak jsou použity v této aplikaci. DWR se skládá ze dvou hlavních částí:

1) Speciální Java servlet, který běží na straně serveru a zpracovává požadavky, volá příslušné Java metody a zasílá zpět odpovědi prohlížeči.
2) Javascriptový kód v prohlížeči, který zasílá požadavky na výše zmíněný servlet a na základě jejich odpovědi aktualizuje HTML stránku.

DWR servlet při spuštění aplikace automaticky vygeneruje javascriptovou reprezentaci Java metod, které mají být volány na straně serveru. Programátor potom v Javascriptu pracuje s těmito metodami a o vše ostatní se již postará DWR. Programátor je tak odstíněn od samotného zasílání AJAX požadavku na server, zpracování jeho odpovědi a převodu Java objektů do jejich javascriptové reprezentace.

V aplikaci jsem DWR s jQuery použil celkem na čtyřech místech:

1) Vytváření, editace a mazání formuláře hovoru na detailu kampaně a přesouvání pořadí jednotlivých otázek v tomto formuláři
2) Přidávání a odebírání operátorů ke kampani
3) Mazání fotografií produktu
4) Zahájení a ukončení telefonního hovoru

Samotnou knihovnu jQuery jsem pak využíval i v jiných částech – prakticky všude, kde se volá Javascript.

Konfigurace DWR servletu je uvedena v souboru *web.xml* a byla již zmíněna v kapitole 5.2.2. Samotná konfigurace knihovny se nachází v souboru *applicationContext-integration.xml* a důležitá jsou následující nastavení.

```
<dwr:configuration>
    <dwr:convert type="bean"
    class="telesystem.entities.callform.CallFormQuestion"/>
    <dwr:convert type="bean" class="telesystem.dto.DWRResultDto"/>
</dwr:configuration>

<dwr:annotation-config />
<dwr:url-mapping />
<dwr:controller id="dwrController" debug="false" />
```

Značka `dwr:convert` definuje konvertor, který určuje, jakým způsobem se budou Java objekty převádět na javascriptové struktury a naopak. DWR nativně podporuje převod všech primitivních typů jazyka Java, jejich objektových reprezentací, kolekcí těchto objektů a DOM objektů. Jakýkoliv uživatelem vytvořený objekt, který bude použit jako parametr nebo návratový typ volání vzdálené metody, tedy musí mít definovaný konvertor. Konvertor typu `bean` projde všechny get a set metody daného objektu a v Javascriptu z nich vytvoří asociativní pole. Značka `dwr:annotation-config` aktivuje konfiguraci vzdálených metod pomocí anotací. `dwr:url-mapping` určuje, že všechny soubory s Javascriptem vygenerované touto knihovnou budou přístupné na URL */dwr/*, odkud je bude uživatel vkládat do jednotlivých stránek, kde se bude DWR používat. A konečně anotace `dwr:controller` definuje speciální kontroler, který slouží jako vstupní bod pro všechny požadavky a dále volá dané vzdálené metody.

Nastavení samotných vzdálených metod může být také uvedeno v XML podobě, ale preferovanějším způsobem jsou Java anotace.

```
@Service("callFormDWRManager")
@RemoteProxy(name = "CallFormDWRManager")
public class CallFormDWRManager {

    @RemoteMethod
    public DWRResultDto addQuestionToCallForm(Long callFormId,
    CallFormQuestion question) {
```

Z hlediska návrhu jsou všechny vzdálené metody umístěny v servisní vrstvě. Anotace `@RemoteProxy` definuje, že tento manažer bude přístupný pro vzdálené volání z Javascriptu pod názvem `CallFormDWRManager`. Každá metoda, která bude vzdáleně volána, pak musí být označena anotací `@RemoteMethod`. Na straně webového klienta je nutné vložit do HTML stránky odkaz na automaticky vygenerované soubory */dwr/engine.js* a */dwr/interface/CallFormDWRManager.js*. Potom už je možné zavolat v Javascriptu metodu `addQuestionToCallForm`, která ve své návratové funkci vrátí reprezentaci objektu `DWRResultDto`, jenž je dále zpracován.

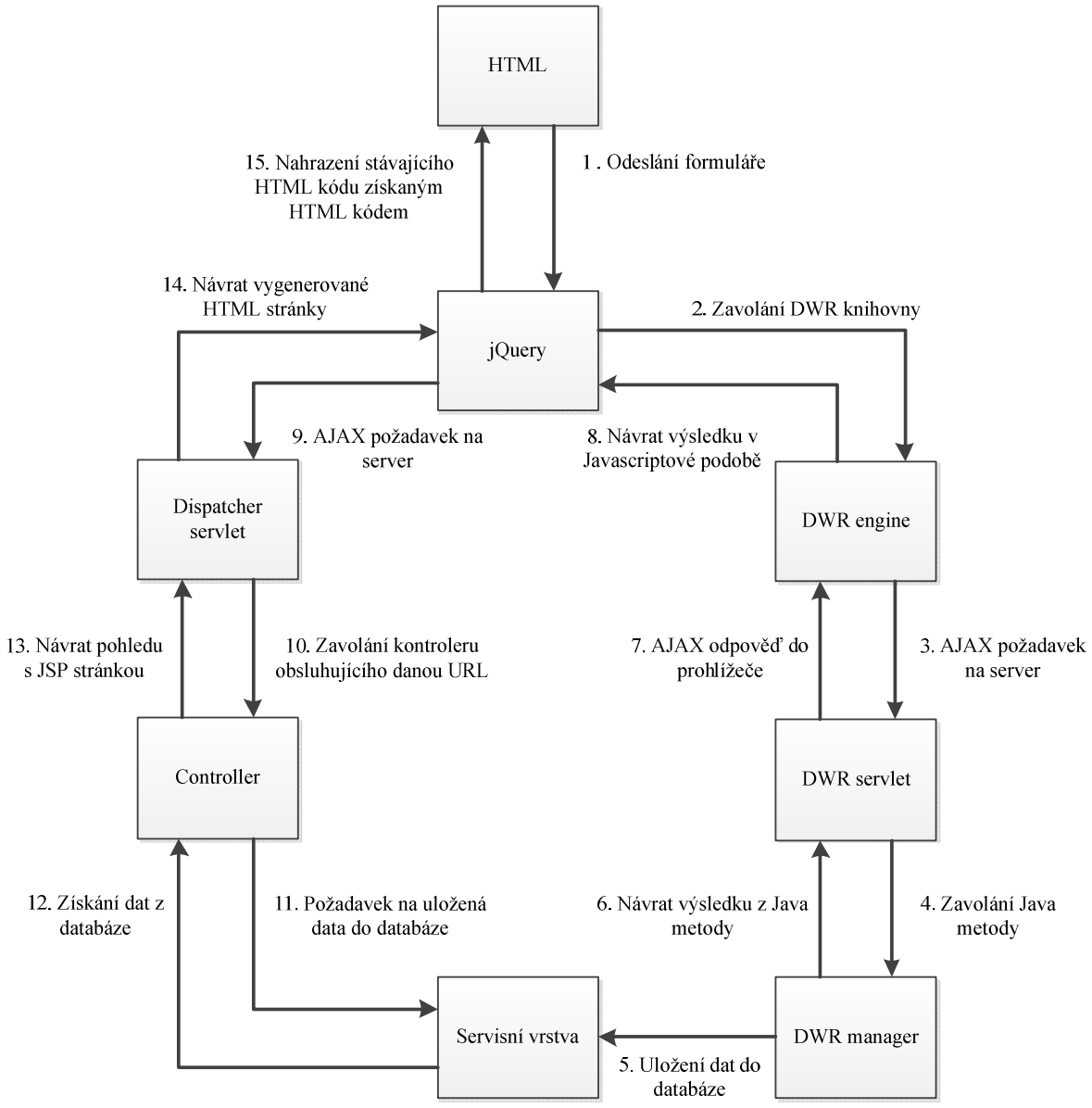
```
CallFormDWRManager.addQuestionToCallForm($("#callform-id").val(),
question, function(data) {
    handleSaveQuestionResult(data);
});
```

Za povšimnutí stojí druhý parametr této metody, kterým je otázka formuláře hovoru. V XML konfiguraci DWR je definován konvertor pro tento typ objektu a tím pádem je možné do tohoto parametru v Javascriptu vložit asociativní pole, kde klíče jsou názvy atributů cílového Java objektu a hodnoty jsou jejich obsah. Podobné je to u návratového objektu, který je naopak naplněn daty v Javě a v Javascriptu převeden na asociativní pole.

Při implementaci přidávání otázek do formuláře hovoru bylo žádoucí, aby se provedené změny ihned reflektovaly v HTML kódu stránky bez jejího znovu načtení a přidané otázky se tedy zobrazily v náhledu formuláře. Toho šlo dosáhnout dvěma způsoby:

1) Při volání vzdálené metody vrátit uloženou otázku, pomocí jQuery sestavit HTML kód pro její zobrazení a tento kód vložit na správnou pozici mezi ostatní otázky (pořadí nově přidané otázky lze libovolně zvolit).
2) Po návratu z volání vzdálené metody zaslat AJAX požadavek na klasický kontroler rámce Spring MVC, který získá celý formulář hovoru v aktuálním stavu z databáze a vrátí JSP stránku, která tento formulář zobrazí. Pomocí jQuery pak získaný HTML kód nahradit za stávající HTML kód.

Rozhodl jsem se pro druhou možnost, která má jednu nevýhodu, ale dvě podstatné výhody. Nevýhodou je bezesporu nutnost zasílat další AJAX požadavek a dotaz do databáze na získání formuláře hovoru v aktuálním stavu. Rychlost těchto operací je však velmi vysoká, a proto tato nevýhoda není příliš podstatná. Mezi výhody patří generování výsledku do JSP stránky, jejíž kód je naprosto stejný jako ten, který zobrazuje formulář hovoru. Pokud by tedy došlo k nějaké změně v zobrazení tohoto formuláře, nemusel by se zároveň upravovat jQuery kód pro zobrazování přidané odpovědi, jako by tomu muselo být v prvním případě. Zároveň se tím dá předejít s problémy zařazení nové otázky na správné místo, jelikož z databáze je celý formulář získán již v seřazené podobě. Celý proces vytváření/editace otázky formuláře hovoru je znázorněn na obrázku 5.4. Správně by na tomto obrázku měla být ještě zobrazena datová vrstva a databáze, jelikož uložení dat do databáze se neděje v servisní vrstvě. Pro tuto kapitolu to však není důležité, a proto je zde zobrazena pouze servisní vrstva.



Obrázek 5.4 Proces vytvoření otázky formuláře hovoru

5.5.3 Tisk do PDF

V požadavcích na implementovaný systém je možnost tisknout formuláře hovoru pro operátory pracující bez PC. Tento požadavek je realizován generováním formulářů hovor do PDF souborů, které lze potom tisknout na tiskárně, případně uložit na disk pro další zpracování. PDF je zkratkou pro Portable Document Format, což je souborový formát vyvinutý firmou Adobe pro ukládání dokumentů nezávisle na hardwaru i softwaru. Pro generování dat z jazyka Java do PDF jsem použil kombinaci knihoven Freemarker a Flying Saucer. V této kapitole bude popsáno, jakým způsobem jsem tyto dvě knihovny zkombinoval a integroval do aplikačního rámce Spring.

Freemarker [17] je šablonovací nástroj pro generování textového (nejčastěji HTML) výstupu. Základem je šablona obsahující speciální značky pro vkládání dynamických dat. Vstupem Freemarker generátoru je tato šablona a potřebná data. Výstupem je potom text osazený dodanými daty. Je to velmi podobné JSP technologii s tím rozdílem, že zde není nutný žádný servlet pro překládání, a proto je Freemarker vhodný pro newebové aplikace nebo pokud generovaný výstup vyžaduje nějaké další zpracování, jako v tomto případě.

Flying Saucer [18] je knihovna, která umožňuje z XML nebo XHTML vstupu za použití CSS stylů vygenerovat PDF soubor nebo obrázek. CSS stylem se definuje vzhled výsledného PDF dokumentu. Knihovna plně podporuje CSS 2.1 a také některé užitečné funkce z CSS 3, což umožňuje vytvářet tyto styly podobným způsobem jako při psaní HTML stránek.

Konfigurace těchto dvou knihoven se nachází v souboru *applicationContext-integration.xml*, kde jsou definovány také ostatní knihovny třetích stran integrované do rámce Spring. Bean s názvem `freemarkerConfig` představuje jádro knihovny Freemarker a obsahuje cestu, kde jsou uloženy šablony s příponou *ftl*. Dále je zde definován bean s názvem `pdfByXhtmlrendererExporter`, který má injektovaný `freemarkerConfig` a také je zde definována cesta k fontům s češtinou. Tento bean je mnou vytvořená třída obsahující metody, které zajišťují generování do PDF. Externí fonty je nutné použít pro správné generování českých textů na různých systémech. Pro vygenerování PDF souboru je nutné zavolat metodu `generatePdf` se dvěma parametry. První z nich je název šablony a druhý je model obsahující data. Vykonání metody se dá rozdělit do následujících částí:

1) Načtení Freemarker šablony se zadaným názvem
2) Vygenerování XHML kódu z načtené šablony a zadaných dat
3) Načtení českých fontů
4) Rozparsování XHML dokumentu.
5) Vytvoření PDF souboru z rozparsovaného XHTML dokumentu pomocí knihovny Flying Saucer
6) Vrácení PDF souboru ve výstupním toku dat

Celý proces získání PDF souboru s formuláři hovoru pro operátory bez PC se pak sestává ze stisknutí odkazu vedoucího na URL, které obsluhuje kontroler pro tisk do PDF. Tento kontroler zavolá metodu servisní vrstvy, která vrátí vygenerovaný PDF soubor ve výstupním toku dat výše popsaným postupem. Tok dat je vložen do HTTP odpovědi a místo pohledu je vrácen prázdný datový typ `null`, což zajistí okamžité vrácení této odpovědi bez dalšího zpracování – generování pohledu. Uživateli se tedy nabídne možnost otevření či stažení výsledného PDF souboru.

5.6 Propojení s telefonní ústřednou

Systém je propojen s telefonní ústřednou Asterisk, která již byla představena v kapitole 5.1. V této kapitole bude podrobněji rozebráno, jakým způsobem je systém s ústřednou propojen, jaké její součásti využívá a jak je ústředna nakonfigurována.

Jednotlivé PC operátorů jsou s ústřednou propojeny pomocí technologie VoIP (Voice over Internet Protocol), která umožňuje přenos digitalizovaného hlasu po klasické počítačové síti a není tedy zapotřebí speciální telefonní linka. Mezi nejznámější protokoly využívající tuto technologii patří H.323, SIP nebo IAX. IAX byl vyvinut primárně pro komunikaci mezi jednotlivými ústřednami Asterisk. H.323 je dnes již starší protokol, který je poměrně složitý, a proto se od něj postupně ustupuje. SIP (Session Initiation Protocol) je jednodušší modernější protokol založený na ověřených principech internetového modelu. Používá se k ustanovení spojení a vlastní přenos hovoru je realizován pomocí protokolu RTP, který se přenáší v těle SIP paketů. Již z podrobnějšího popisu protokolu SIP je zřejmé, že jsem pro komunikaci operátorů s telefonní ústřednou vybral právě tento protokol. Předpokládá se, že každý PC operátora bude vybaven softwarovým telefonem pracujícím na protokolu SIP a sluchátky s mikrofonem. Na přiloženém DVD je obraz virtuálního PC s operačním

systémem Windows XP, který reprezentuje prototyp PC pro operátora telemarketingové společnosti a je tedy vybaven tímto softwarovým telefonem.

Z požadavků na systém vyplynuly tři situace, ve kterých musí být využita telefonní ústředna:

1) Spojení telefonního hovoru operátora pracujícího na PC s vybraným telefonním kontaktem
2) Nahrávání průběhu každého hovoru a ukládání jeho technických informací
3) Získávání informací o hovoru a možnost přehrávání jeho záznamu

Tyto tři požadavky jsou v systému splněny. První požadavek však není splněn doslovně, jelikož telefonní ústředna není propojena s venkovní telefonní sítí PSTN (Public Switched Telephone Network). Pro propojení s touto sítí je zapotřebí speciální hardware, který jsem neměl v průběhu vytváření aplikace k dispozici. Veškeré telefonní hovory jsou tedy směrovány na testovací softwarový telefon, který je připojen pomocí protokolu SIP stejně jako všichni operátoři. Podrobnější informace o připojení telefonů operátorů budou poskytnuty v následující kapitole.

5.6.1 Konfigurace ústředny Asterisk

Asterisk je od začátku vyvíjen primárně pro operační systém Linux, jelikož ten je nejvhodnější pro serverové systémy. I já jsem se tedy rozhodl, že telefonní ústředna tohoto systému poběží na operačním systému Linux. Při instalaci ústředny Asterisk je možné postupovat několika způsoby. Nejvíce konfigurovatelný přistup je vlastní instalace vybrané distribuce OS linux, stažení potřebných zdrojových souborů pro Asterisk a dalších potřebných nástrojů a jejich následná kompilace. Samotný Asterisk však nabízí předinstalovaný operační systém i s telefonní ústřednou a potřebnými nástroji v podobě balíku AsteriskNOW [27]. Zvolil jsem právě tuto možnost a použil jsem AsteriskNOW verze 1.7.1, která obsahuje operační systém CentOS verze 5.5, Asterisk 1.6.2 a zejména program FreePBX verze 2.7.0. Asterisk se konfiguruje pomocí různých konfiguračních souborů, ve kterých je někdy velmi náročné vyhledat to správné nastavení. FreePBX je webové uživatelské rozhraní, které tento problém částečně řeší a umožňuje konfigurovat většinu nastavení Asterisku pohodlněji pomocí webových formulářů. Lze k němu přistoupit na IP adrese telefonní ústředny v protokolu HTTP.

Po spuštění nainstalovaného balíčku AsteriskNOW byly všechny nástroje nastaveny a spuštěny a bylo tedy možné začít konfigurovat propojení se systémem telemarketingové společnosti. Pro každého nového operátora je potřeba vytvořit unikátní telefonní klapku, přes kterou se bude připojovat k ústředně při provádění telefonních hovorů. V systému FreePBX je tedy potřeba zvolit menu *Extensions* a ve výběrovém poli *Device* zvolit *Generic SIP Device* a odeslat formulář. Zobrazí se nový formulář pro vytvoření nové klapky, do kterého je potřeba vyplnit minimálně následující údaje:

- User Extension – číslo klapky, unikátní číslo v rámci ústředny
- Display Name – jméno, které se bude zobrazovat na telefonu
- Outbound CID – číslo, které se bude zobrazovat mimo síť ústředny
- Secret – heslo pro přihlašování k ústředně
- Record Outgoing – nastavení nahrávání odchozích hovorů, je nutné nastavit na hodnotu *Always* pro nahrávání všech telefonních hovorů provedených tímto operátorem, více o tomto nastavení bude zmíněno v kapitole 5.6.3

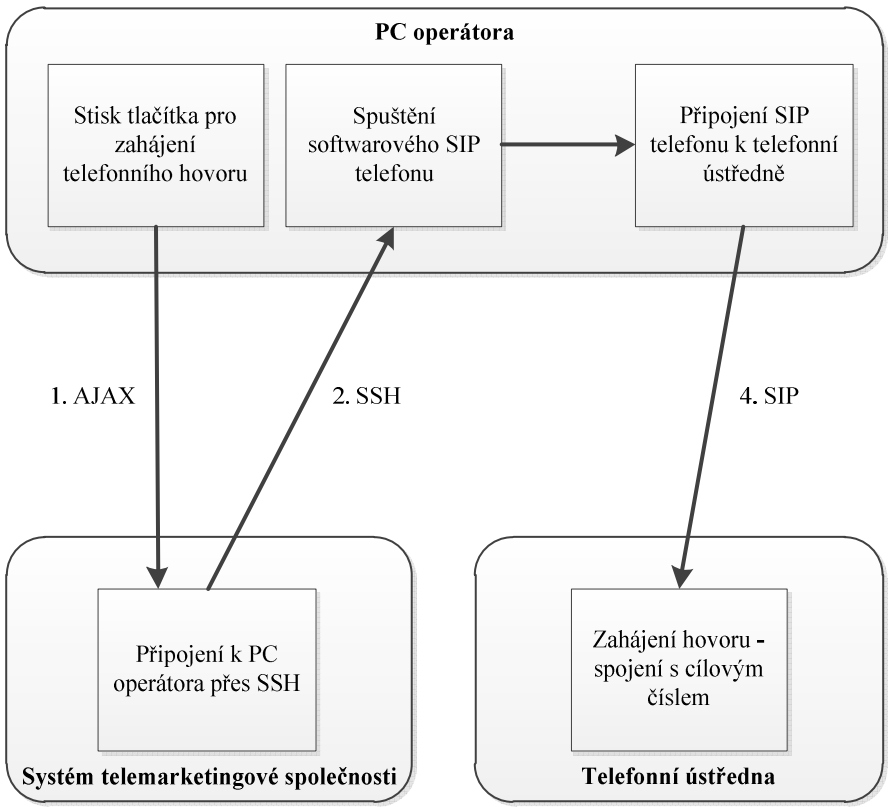
Ostatní údaje doporučuji ponechat tak, jak jsou předvoleny. Po úspěšném vytvoření nové klapky je možné vytvořit operátorovi uživatelský účet v samotném systému a do sekce „Připojení

k telefonní ústředně“ zadat ID a heslo, kde ID je hodnota zadaná v poli *User Extension* a heslo je hodnota z pole *Secret* zadané v systému FreePBX.

5.6.2 Provádění telefonních hovorů

V této kapitole bude popsáno, jakým způsobem jsou prováděny telefonní hovory operátorů pracujících na PC pomocí tohoto systému. Jak již bylo zmíněno, tak každý operátor bude mít na svém PC softwarový telefon pracující na protokolu SIP. Požadavkem na systém je spuštění telefonního hovoru stiskem tlačítka na detailu hovoru. Technicky však není možné, aby stiskem tlačítka v HTML stránce ve webovém prohlížeči byl spuštěn nějaký program na daném operačním systému. Uživatel může přes webový prohlížeč komunikovat pouze se systémem telemarketingové společnosti. Tento problém jsem v aplikaci vyřešil níže popsaným způsobem.

Operátor v prohlížeči stiskne tlačítko pro zahájení hovoru a pomocí DWR a jQuery se odešle asynchronní požadavek na server. Server se připojí přes protokol SSH k počítači operátora a spustí softwarový telefon. Následně se tento telefon připojí k telefonní ústředně pomocí protokolu SIP a spustí hovor. Telefonní ústředna se spojí s požadovaným cílovým číslem a začne zprostředkovávat přenos samotného hovoru. Celý proces je znázorněn na obrázku 5.5.



Obrázek 5.5 Zahájení telefonního hovoru

Server se spojí s počítačem operátora pomocí protokolu SSH. Jedná se o zabezpečený komunikační protokol umožňující komunikaci mezi dvěma počítači. Slovo zabezpečený napovídá, že před ustanovením spojení musí proběhnout autentizace. Existují dva způsoby, jak se autentizovat. Prvním z nich je použití přihlašovacího jména a hesla. Druhým způsobem je použití privátního klíče na klientské straně a veřejného klíče na straně serveru. Využil jsem první způsob a přihlašovací jméno i heslo je uloženo v konfiguračním souboru *config.properties*. Jak jsem již zmínil výše, tak

prototyp PC pro operátora je vybaven operačním systémem Windows XP, který však žádný SSH server nativně neobsahuje. Použil jsem tedy program freeSSHd [28], který je nainstalovaný na prototypovém PC a je spouštěn automaticky po startu systému. Pro připojení ze strany systému telemarketingové společnosti se nabízely dvě možnosti. První bylo použití nativního SSH klienta OpenSSH v linuxové distribuci CentOS a ovládat ho zasíláním příkazů z Javy nebo použít některou Java knihovnu implementující SSH klienta a připojovat se tak přímo v Java kódu. Druhá možnost je jistě lepší volbou, a tak jsem využil knihovnu sshj [29] poskytující podobné možnosti jako program OpenSSH.

Po připojení přes SSH je klientskému stroji (v tomto případě server se systémem společnosti) umožněn přístup do konzole operačního systému. Většina softwarových SIP telefonů se ovládá přes grafické uživatelské rozhraní a není tedy uzpůsobena pro ovládání přes konzolu. Pro tento účel je na prototypovém PC nainstalovaný program s názvem pjsua [30]. Jedná se o softwarový SIP telefon, který je primárně určený pro ovládání pomocí konzole systému. Je možné jej provozovat jak na operačních systémech linux, tak Windows. Server tak postupně posílá příkazy na počítač operátora, které ovládají softwarový telefon. Nejprve se spustí příkaz pro připojení k telefonní ústředně.

```
pjsua.exe --id sip:{0}@{1} --registrar  sip:{1} --realm * --username {0}
--password {2}\r
```

Složené závorky představují proměnné, za které budou nahrazeny následující parametry označené daným číslem:

0) SIP identifikátor – unikátního číslo klapky operátora
1) IP adresa telefonní ústředny
2) Heslo operátora pro připojení k telefonní ústředně

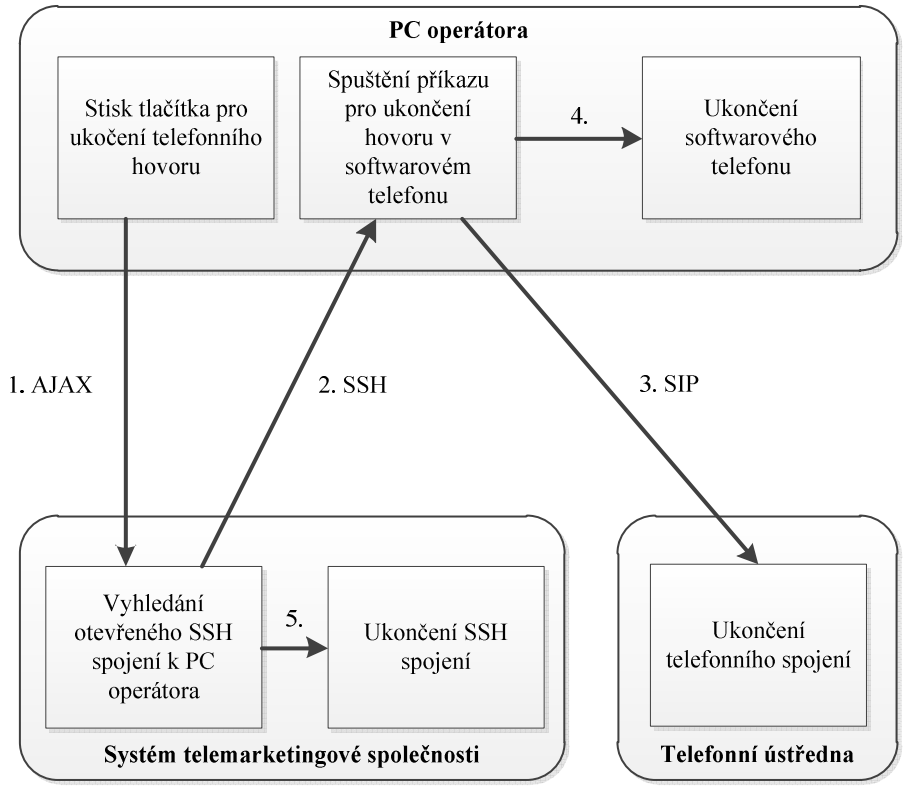
Po úspěšném připojení následuje příkaz `m\r` pro spuštění nového hovoru a po něm příkaz pro zadání cílového telefonního čísla.

```
sip:{0}@{1}\r
```

Za první složenou závorku je doplněno volané telefonní číslo, za druhou potom opět IP adresa ústředny. Po spuštění tohoto příkazu je zahájen samotný hovor. Úplný výčet příkazů, které program pjsua poskytuje, je možné nalézt na webových stránkách tohoto projektu [30].

Na obrázku 5.5 jsou znázorněny celkem 3 prostředí – PC operátora, server, na kterém běží systém telemarketingové společnosti a telefonní ústředna. Na přiloženém DVD je systém společnosti z praktických důvodů nasazen na stejném virtuálním PC jako telefonní ústředna. Obě prostředí však mohou běžet samostatně.

Ukončení telefonního hovoru probíhá podobným způsobem jako jeho zahájení. Tento proces je zobrazen na obrázku 5.6. Po zahájení hovoru je SSH spojení uchováno na serveru pod číslem klapky operátora. Je totiž možné, že v současné době bude více otevřených spojení najednou a tímto způsobem jsou potom jednoznačně odlišena. Při ukončování hovoru je tedy toto spojení vyhledáno a jsou odeslány dva příkazy do softwarového telefonu. Prvním je příkaz `h\r` pro zavěšení hovoru a druhým `q\r` pro ukončení softwarového telefonu. Následně je SSH spojení s počítačem operátora ukončeno a není tedy již nadále udržováno na serveru.



Obrázek 5.6 Ukončení telefonního hovoru

Při procesu provádění telefonních hovorů může nastat několik neobvyklých situací, které nejsou z časových důvodů součástí řešení této práce a možné řešení některých z nich je zmíněno v závěru práce. Seznam nedořešených situací je zobrazen v tabulce 5.1 spolu se současnou reakcí vyvíjeného systému.

Nedořešená situace	Reakce/řešení
Nelze se připojit k telefonní ústředně	Je zobrazena hláška „*Nepodařilo se spustit hovor, prosím zkuste to znovu*“
Spojení s ústřednou je v průběhu hovoru přerušeno	Operátor musí stisknout tlačítko pro ukončení hovoru
Volaný položí hovor	Operátor musí stisknout tlačítko pro ukončení hovoru
Při ukončování hovoru již není spuštěný softwarový telefon na PC operátora	Systém se chová stejně jako by ho ukončil a zobrazí tedy hlášku „*Hovor byl úspěšně ukončen*“
Při ukončování hovoru již není otevřené SSH spojení s PC operátora	Je zobrazena hláška „*Nepodařilo se ukončit hovor. Hovor byl pravděpodobně ukončen již dříve*“

Tabulka 5.1 Řešení neobvyklých situací při provádění telefonního hovoru

Teoreticky by mohla ještě nastat situace, kdy operátor zahájí nový telefonní hovor, ale ještě není ukončeno SSH spojení pro předchozí hovor. To by však nemělo nastat, jelikož systém nezobrazí tlačítko pro zahájení hovoru, dokud není původní hovor ukončen. Kvůli takové možnosti jsou ale preventivně před zahájením každého nového SSH spojení všechna otevřená spojení daného operátora

uzavřena. Tím je zajištěno, že pro každého operátora může být souběžně otevřeno maximálně jedno SSH spojení.

5.6.3 Zaznamenávání telefonních hovorů

Telefonní ústředna Asterisk umožňuje jak ukládání technických informací, tak ukládání audio záznamu o každém provedeném hovoru. V této kapitole bude popsáno, jakým způsobem využívá těchto vlastností implementovaný systém.

Call Detail Recording

Funkcionalita ukládající technické informace o hovorech se v ústředně Asterisk nazývá Call Detail Recording (CDR). Implicitně tato data Asterisk ukládá ve formátu CSV (Comma Separated Values). Poskytuje však také podporu pro ukládání do různých relačních databázových systémů. Balíček AsteriskNow má přednastavené ukládání CDR do databáze MySQL. Toto nastavení jsem ponechal. Připojovací údaje, název databázové tabulky a další konfigurační údaje ukládání CDR se nachází v souboru *cdr_mysql_conf* v adresáři */etc/asterisk*. CDR se ukládá do jedné databázové tabulky, která obsahuje následující sloupce:

- **calldate** – datum a přesný čas zahájení hovoru
- **clid** – název volajícího, v tomto případě hodnota vyplněná v atributu *Display name* v systému FreePBX u daného operátora
- **src** – ID volajícího, v tomto případě SIP klapka operátora
- **dst** – číslo volaného
- **dcontext** – cílová doména
- **channel** – použitý kanál, v tomto případě SIP kanál
- **dstchannel** – cílový kanál, v tomto případě SIP kanál
- **lastapp** – poslední použité pravidlo v plánu hovoru
- **lastdata** – poslední použité nastavení v plánu hovoru
- **duration** – celkový čas hovoru
- **billsec** – čistý čas hovoru mezi zvednutím a položením volaného
- **disposition** – výsledek hovoru, celkem 4 možnosti – zvednuto, nezvednuto, obsazeno, nezdařeno
- **amaflags** – Automated Message Accounting příznaky, jsou zde spíše z historického hlediska, celkem 3 možnosti – nenahrávat hovory, ukládat hovor pro vyúčtování, ukládat hovor pro dokumentaci
- **accountcode** – číslo účtu pro vyúčtování hovoru, v tomto případě bude prázdné
- **uniqueid** – unikátní identifikátor v rámci dané CDR databáze
- **userfield** – uživatelem definované informace

Přestože bylo po instalaci balíčku AsteriskNow vše řádně nastaveno, ústředna nezaznamenávala žádné informace o hovorech. Později jsem zjistil, že některé komponenty nutné pro ukládání CDR informací již nejsou z licenčních důvodů součástí balíčku AsteriskNOW, ale jsou součástí Asterisk doplňků. Bylo tedy nutné stáhnout tyto doplňky příkazem `yum install asterisk16-addons` a po úspěšném stažení zastavit a následně znovu spustit ústřednu Asterisk pomocí příkazů `amportal stop` a `amportal start`. Po tomto procesu již ústředna ukládala informace o každém provedeném hovoru.

Při zobrazování detailu některého provedeného telefonního hovoru ve vyvíjeném systému se tento připojí k databázi CDR a podle atributu *uniqueid* získá záznam s informacemi o daném hovoru. Tyto informace jsou potom zobrazeny spolu s klasickými informacemi o hovoru ze systému. Systém tak v tomto momentu získává data ze dvou rozdílných databází. Způsob konfigurace tohoto přístupu je stručně popsaný již v kapitole 5.3.2. Nyní však vyvstává otázka, jak získá systém atribut *uniqueid* u daného hovoru. Tento atribut je uložen v tabulce *phone_call* v databázi systému a získává se ihned po provedení daného hovoru zavoláním dotazu do databáze CDR na získání posledního hovoru provedeného pod klapkou daného operátora. Společně s atributem *uniqueid* je získán ještě čistý čas hovoru v atributu *billsec*. Ten se využívá při zobrazování statistik hovorů pro výpočet průměrné doby hovoru.

Audio záznamy

Asterisk umožňuje jako každá ústředna nahrávání audio záznamů. Tuto funkcionalitu zajišťuje modul Monitor. Ve webovém rozhraní FreePBX se nahrávání nastavuje u každé klapky zvlášť v sekci *Recording Options*. Lze zde nastavit nahrávání příchozích a odchozích hovorů ve třech režimech:

- **On Demand** – nahrávat se začne pouze po spuštění speciálního kódu v průběhu hovoru. Implicitně se jedná o kód **1*. Tento kód se dá nastavit v menu *Feature Codes* v sekci *Core*. Pro zprovoznění tohoto typu nahrávání je ještě nutné v menu *General Settings* v sekci *Dialing Options* nastavit příkaz pro příchozí resp. odchozí hovory s parametrem *w* nebo *W*. Parametr *w* slouží k povolení možnosti zapnout nahrávání z telefonu volaného, parametr *W* potom z telefonu volajícího.
- **Always** – nahrávat se bude každý hovor. V případě systému telemarketingové společnosti by tato volba měla být nastavena na klapce každého operátora.
- **Never** – žádný hovor se nebude nahrávat.

Další obecné nastavení nahrávání se nachází v menu *General Settings* v sekci *Call Recording*. Je zde možné globálně vypnout nahrávání pro všechny klapky, vybrat formát audio záznamů nebo změnit složku pro ukládání nahrávek. V této práci je nastaven audio formát *wav* a složka pro nahrávání je ponechána na přednastavenou */var/spool/asterisk/monitor/*.

Ve vyvíjeném systému se audio záznamy zobrazují na detailu provedeného hovoru spolu s informacemi z telefonní ústředny. Záznamy jsou zde uvedeny ve formátu HTML5 a je tedy možné je přehrávat přímo v prohlížeči, případně uložit na disk k dalšímu zpracování.

```
<c:set var="recordFileUrl" value="/detail-
hovoru/${model.call.id}/zaznam/${model.cdr.audioRecordFileName}"/>
<audio src="<c:url value="${recordFileUrl}"/>" controls="controls">
    <a href="<c:url value="${recordFileUrl}"/>">stáhnout</a>
</audio>
```

HTML5 značka `audio` pro přehráváním souboru ve formátu *wav* je aktuálně podporována v prohlížečích Firefox 3.6, Google Chrome 11, Opera 11, Internet Explorer 9, Safari 5.0 a některých dalších. Předpokládá se, že všechny PC v telemarketingové společnosti budou vybaveny prohlížečem Mozilla Firefox 4.0.1 stejně jako prototypový PC operátora, a tudíž by přehrávání mělo fungovat bez problémů. Pokud se však detail provedeného hovoru zobrazí na starším prohlížeči nepodporujícím HTML5, tak se místo přehrávače zobrazí odkaz pro stažení audio záznamu. Samotný soubor se záznamem je do prohlížeče dodáván z URL, kterou obsluhuje kontroler pro získávání audio záznamů. Tento kontroler získá záznam s názvem předaným v parametru ze složky, která je nakonfigurována v souboru *config.properties*. Všechny záznamy jsou v této složce ukládány pod názvem se syntaxí

`IN{0}-{1}-{2}.wav`, kde čísla ve složených závorkách jsou zastoupena následujícími hodnotami z databáze CDR:

0) Číslo volaného (dst)
1) Datum a přesný čas začátku hovoru ve formátu yyyyMMdd-HHmmss (calldate)
2) Unikátní identifikátor (uniqueid)

Název audio záznamu může mít tedy například následující podobu: IN200-20110420-132751-1303298871.3.wav.

5.7 Optimalizace výkonnosti

Jedním z důležitých požadavků na systém je jeho vysoká výkonnost. Pokud by systém při velkém zatížení reagoval na požadavky uživatelů velmi pomalu, práce s ním by se stávala nepříjemnou a zejména by mohly společnosti unikat zisky z času stráveného čekáním na odpověď. Operátoři by museli dlouho čekat při ukládání vyplněných a načítání nových systémových hovorů a nemohli by se věnovat telefonování. Tato kapitola se bude zabývat optimalizací systému z hlediska výkonnosti pro poskytování rychlých odpovědí i při velkém zatížení.

5.7.1 Testovací data a měření výkonu

Při reálném provozu je vcelku snadné měřit výkonnost systému, jelikož ten je již naplněn reálnými daty, nad nimiž probíhají reálné operace. Pomalá místa jsou potom odhalena buď samotným používáním, nebo posbíráním statistických údajů o průběhu jednotlivých operací. V systému, který ještě nebyl nasazen do skutečného provozu, je situace složitější. Většinou nejsou k dispozici skutečná data a není jisté, které operace budou s jakou frekvencí používány. V takovém případě je potřeba provést analýzu budoucího využití aplikace a vytvořit sadu testovacích dat a měřících procedur, které budou reflektovat skutečný provoz a odhalí úzká místa z hlediska výkonnosti. Dalším problémem v doposud nenasazené aplikaci je neznalost nebo nemožnost otestování na hardware, na kterém bude skutečně provozována. Velmi pomalá místa se však odhalí i při testování na jiném hardware.

Při analýze vyvíjeného systému telemarketingové společnosti jsem dospěl k názoru, že úzkým místem bude jistě relační databáze, jelikož systém je silně databázově orientovaný a téměř každý požadavek provedený uživatelem vyžaduje přístup do databáze. Při testování pouze s iniciálními daty v databázi je odezva systému rychlá a práce s ním plynulá. Je ovšem potřeba systém otestovat, jak se bude chovat s větším vzorkem dat. Tabulky uchovávající data o provedených telefonních hovorech a tabulky s telefonními kontakty budou zřejmě ty s nejrychleji rostoucím počtem záznamů. Vytvořil jsem tedy skript pro naplnění databáze 100 000 telefonními kontakty, 50 000 provedenými hovory pro jeden produkt, 50 000 provedenými hovory pro jeden průzkum a 1000 operátory. Původně jsem chtěl vygenerovat 1 000 000 provedených hovorů, ale to bylo časově příliš náročné a navíc jsem si ověřil, že testovaný vzorek je dostatečně průkazný. Skript je přiložený ve složce *src/database* pod názvem *create-test-data.sql*.

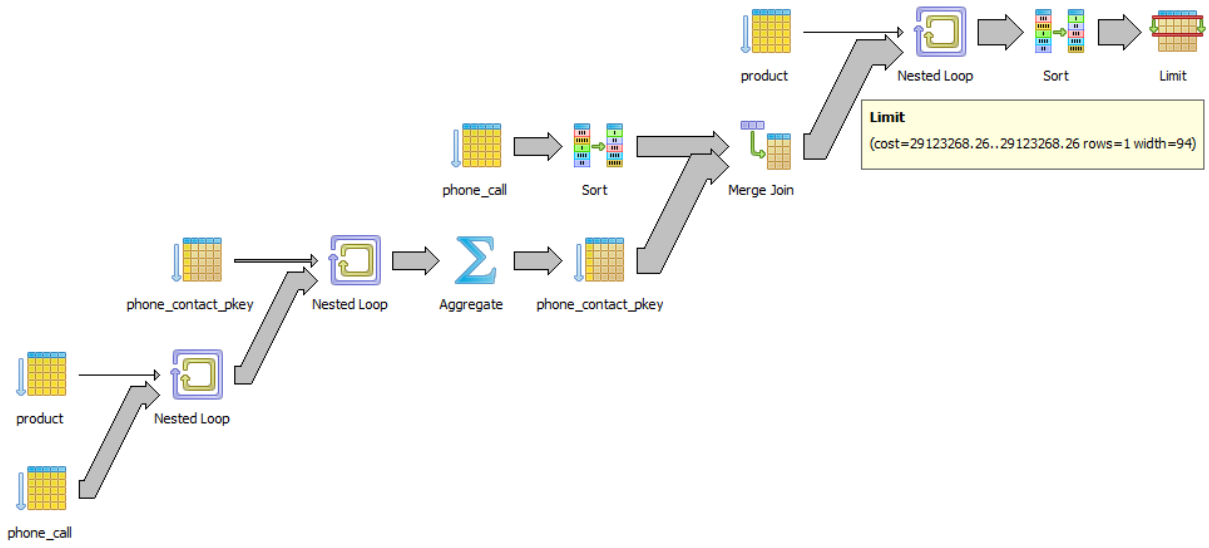
Nejvyšší výkonnostní požadavky jsou kladeny na vybírání telefonních kontaktů pro nový systémový hovor operátora pracujícího na PC. Tato funkcionalita je zároveň nejsložitější z databázového pohledu, jelikož se může skládat až ze čtyř složitých databázových dotazů za sebou, které vyhledávají nad potenciálně velmi obsáhlými tabulkami. V následující části tedy budu popisovat proces měření výkonu a případné optimalizace této funkcionality. Skript pro naplnění dat je záměrně vytvořen tak, aby pro každý telefonní kontakt vytvořil jeden provedený hovor. Tím je zaručeno, že při

vybírání nového telefonního kontaktu pro operátora proběhnou všechny čtyři databázové dotazy zobrazené na diagramu 5.1. Testování jsem prováděl na procesoru Intel Core 2 Duo T8100 2.2 GHz se 4 GB DDR2 RAM. Systém byl spuštěn na aplikačním serveru Apache Tomcat 6.0 a databázi PostgreSQL. Obě tyto technologie jsou představeny v kapitole 5.1.

Měření výkonu

Pro simulaci zátěže a měření výkonu jsem použil open source nástroj JMeter [31] vyvinutý společností Apache Foundation. Jedná se desktopovou aplikaci implementovanou v jazyce Java, která dokáže zasílat různé typy požadavků s různou frekvencí ve více vláknech současně. Umožňuje tak simulovat více uživatelů najednou a přehledně měřit výsledky. V tomto nástroji jsem vytvořil jednoduchý test, který odesílá HTTP požadavek metodou GET na URL */hovor* a následně na tu samou URL odesílá HTTP požadavek metodou POST s vyplněnými povinnými parametry formuláře hovoru. Tento test tedy simuluje operátora pracujícího na PC, kterému se po přihlášení zobrazí tato stránka. Operátor provede telefonní hovor, zapíše výsledek do formuláře a uloží jej. Systém mu vygeneruje nový formulář hovoru pro další telefonní kontakt a takto to pokračuje stále dokola. Kvůli úspěšné simulaci více operátorů současně jsem musel udělat drobné úpravy v aplikaci. Musel jsem povolit tuto URL bez přihlašování a operátora rozpoznávat podle parametru id v URL. Obsah tohoto parametru se generuje automaticky ve vytvořeném testu. Pokud např. test spustím s deseti operátory současně, dosadí se do tohoto parametru hodnoty 1 až 10. Vytvořený test se nachází ve složce *jmeter/* pod názvem *operator.jmx*.

Po naplnění databáze testovacími daty jsem nejprve spustil tento test pouze s jedním operátorem. Dočkal jsem se však velmi neblahého výsledku, jelikož požadavek na vygenerování nového systémového hovoru se nedokončil a stále se prováděl. Jelikož byl procesor vytížen procesem databázového stroje, je velmi pravděpodobné, že byl prováděn některý náročný databázový dotaz. Z logu PostgreSQL jsem zjistil, že systém vytížil poslední SQL dotaz na získání kontaktu s nejstarším provedeným hovorem v aktuálním kole obvolávání. Spuštěním tohoto dotazu s příkazem *EXPLAIN* lze zobrazit plán provádění a také časovou náročnost jednotlivých procesů dotazu. V programu pgAdmin III, který je součástí instalace PostgreSQL pro Windows, lze tento plán zobrazit také v interaktivní grafické podobě, která je mnohem přehlednější. Tento grafický plán je zobrazen na obrázku 5.7.

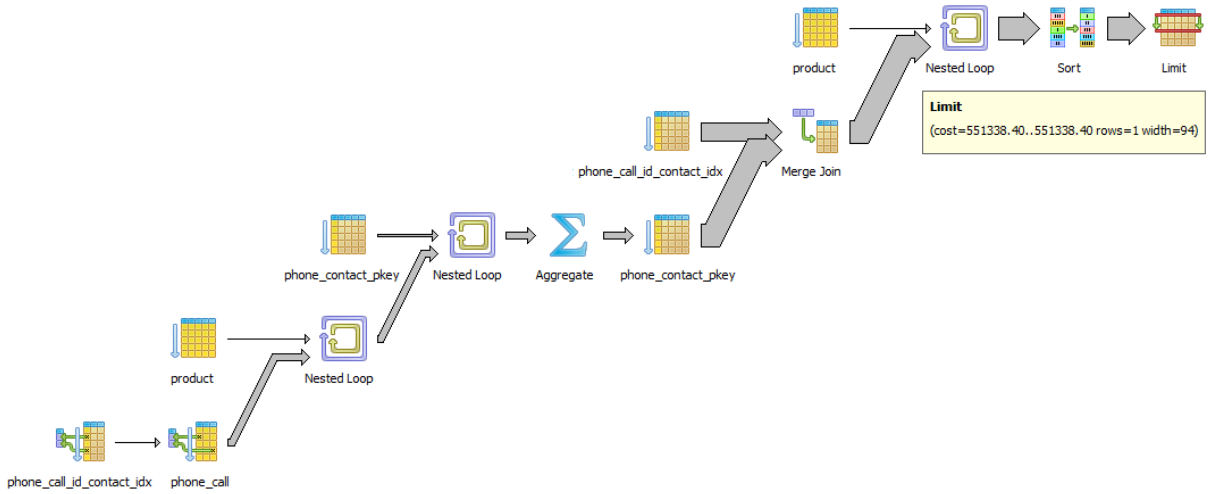


Obrázek 5.7 Plán provádění náročného SQL dotazu

Výsledkem dotazu je časová náročnost cost=29123268.26..29123268.26. Tento údaj je odhad plánovače databázového stroje, jak dlouho bude trvat vykonání dotazu. Je uveden v jednotkách vytažených diskových stránek. Jsou zde uvedena vždy dvě čísla. První reprezentuje čas, než může být vrácen první řádek. Druhé číslo reprezentuje celkový čas, za který budou vráceny všechny řádky. V tomto případě jsou obě čísla stejná, jelikož je výsledek předem omezen na jeden vrácený řádek. Přestože je tento údaj pouze orientační, měl by se u často používaného dotazu pohybovat maximálně v řádech tisíců. Náročnost tohoto dotazu na testovacích datech se pohybuje v řádech desítek milionů, což je při „pouhých“ 100 000 položek v prohledávaných tabulkách nepřípustně pomalé. V tento moment tedy musel nastat proces optimalizace tohoto dotazu.

Optimalizace náročného SQL dotazu

Žlutě vybarvené tabulky na obrázku 5.7 reprezentují sekvenční čtení nad danou tabulkou. Sekvenční čtení je při velké množství dat nejpomalejším typem čtení, jelikož se prochází celá tabulka řádek po řádku, dokud není nalezena požadovaná hodnota. Vhodnou optimalizací je použití indexu nad čtenými sloupci. Vytvořil jsem tedy index metodou btree nad sloupcem *id_contact* v tabulce *phone_call*. Plán dotazu po aplikaci indexu je zobrazen na obrázku 5.8.

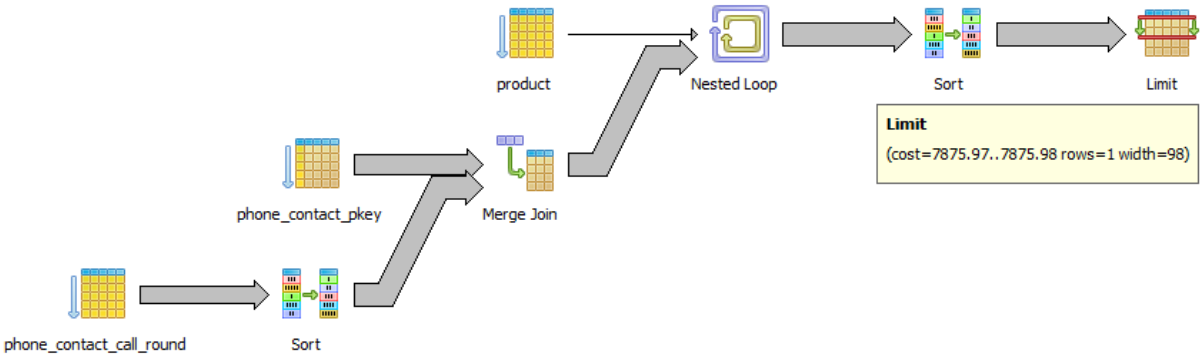


Obrázek 5.8 Plán provádění náročného SQL dotazu po aplikaci indexu

Relativní náročnost se přibližně 50krát snížila, což je výborný výsledek. Stále se však pohybuje v řádech stovek tisíců. Poslední sekvenční čtení probíhá nad tabulkou *product*. Zde však již není možné použít index, jelikož v této tabulce je pouze jeden záznam a plánovač tedy vždy aplikuje sekvenční čtení, protože bude rychlejší. Po spuštění výše zmiňovaného testu v programu JMeter pro jednoho operátora již oba požadavky skončí v celkovém čase necelých 5 sekund. Tento čas je pod 10 sekund, a tudíž je přijatelný z hlediska požadavků na systém. Při spuštění daného testu pro 10 operátorů současně se však průměrný čas zpracování obou požadavků zvýší na přibližně 21 sekund, což už není přijatelné. Nejnáročnější částí dotazu je subdotaz pro získání informace o tom, zda danému kontaktu již bylo voláno Xkrát, kde X reprezentuje aktuální kolo obvolávání. Tento údaj by se mohl ukládat u každého telefonního kontaktu do nového sloupce, aby se nemusel aplikovat subdotaz na jeho zjištění. Jelikož však jeden telefonní seznam může být přiřazen k více produktům nebo průzkumům, tak by bylo potřeba vytvořit pomocnou tabulku uchovávající odkaz na kontakt, produkt, průzkum, datum a čas posledního volání a aktuální kolo obvolávání. Tato tabulka by se

mohla aktualizovat při každém uložení provedeného hovoru a náročný SQL dotaz by mohl být nahrazen dotazem pouze nad touto tabulkou bez dalších subdotazů.

Na obrázku 5.9 je zobrazen plán provádění SQL dotazu nad nově vytvořenou pomocnou tabulkou *phone_contact_call_round*.



Obrázek 5.9 Plán provádění optimalizovaného SQL dotazu

Na první pohled je dotaz jednodušší a jeho relativní časová náročnost je v řádech tisíců, což je oproti původnímu dotazu vynikající. Po spuštění testu pro 10 operátorů současně skončily oba požadavky v průměru do 3 sekund, pro 30 operátorů současně do 9 sekund. Tyto výsledky z programu JMeter jsou zobrazeny na obrázku 5.10 a 5.11.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
hovor	10	814	125	1356	398,19	0,00%	5,0/sec	73,04	15033,0
ulozit hovor	10	1783	747	3121	877,70	0,00%	2,2/sec	31,90	15033,0
TOTAL	20	1298	125	3121	836,21	0,00%	4,2/sec	62,10	15033,0

Obrázek 5.10 Výsledky testu pro 10 operátorů současně

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
hovor	30	2804	133	12498	2123,85	0,00%	2,3/sec	34,13	15033,0
ulozit hovor	30	5830	962	10538	2678,06	0,00%	2,1/sec	31,34	15033,0
TOTAL	60	4317	133	12498	2851,41	0,00%	4,2/sec	62,10	15033,0

Obrázek 5.11 Výsledky testu pro 30 operátorů současně

Sloupec *Average* udává průměrnou dobu odezvy v milisekundách. Naměřené hodnoty se dají považovat za přijatelné. Původní neoptimální dotaz jsem ve zdrojovém kódu ponechal okomentovaný. Metoda a třída, ve které se nachází, je zmíněna v kapitole 5.3.4.

Podobným způsobem by bylo možné otestovat a případně optimalizovat i ostatní databázové dotazy, avšak to jednak není součástí této práce a jednak jsou ostatní dotazy daleko jednodušší. Neměly by tedy působit výkonnostní problémy a bude u nich postačovat pouze zapnutí vyrovnávací paměti rámce Hibernate.

5.7.2 Hibernate cache

Jednou z možností, jak snížit zátěž databázového systému, je využití vyrovnávací paměti. Do této paměti je možné ukládat výsledky často používaných databázových dotazů a při dalším požadavku na jejich získání tak nezasílat nový dotaz do databáze, ale pouze vrátit tento výsledek z vyrovnávací paměti. Vyrovnávací paměť je uložena v operační paměti počítače a je tedy velmi rychlá. Rámec Hibernate poskytuje dva typy vyrovnávací paměti – tzv. cache.

- **First-level cache** – udržuje načtené objekty a kolekce v rámci jedné session, je zapnuta vždy a nelze ji vypnout, není sdílena mezi jednotlivými transakcemi
- **Second-level cache** – standardně je vypnuta, je sdílená mezi transakcemi, je to uživatelsky konfigurovatelná cache.

First-level cache je tedy zapnuta vždy a vývojář ji nemusí nijak konfigurovat. Její příspěvek ke zvýšení výkonu aplikace je však znatelný pouze, pokud jsou v průběhu jedné transakce opakovaně získávány z databáze stejné objekty. V případě vyvíjeného systému však k této situaci příliš nedochází. Second-level cache již není implementována samotným rámcem Hibernate. Ten pro ni pouze poskytuje podporu a pro implementaci lze využít některou z knihoven třetích stran, jako je například Ehcache, JBoss TreeCache, SwarmCache nebo OSCache. Pro tento systém jsem zvolil knihovnu Ehcache [32] od společnosti Terracotta. Second-level cache můžeme rozdělit na tři typy podle způsobu jejího použití:

- **Cache atributů** – definuje se pro jednotlivé perzistentní entity a tyto jsou potom v této cache uloženy v asociativní tabulce, kde klíčem je primární klíč objektu a hodnotou je další asociativní tabulka obsahující názvy atributů a jejich hodnoty. Nejsou zde tedy uchovávány celé instance těchto objektů.
- **Cache relací** – definuje se pro relace v perzistentních třídách a obsahuje primární klíče všech objektů v dané relaci.
- **Cache výsledků dotazů** – definuje se pro jednotlivé databázové dotazy. Obsahuje podobu daného dotazu s parametry, se kterými byl volán a primární klíče objektů vrácených jako výsledek.

V aplikaci jsem pro maximální možné zvýšení výkonu zkombinoval tyto tři typy dohromady, čímž jsem dosáhl následujícího chování. Při opakovaném volání stejného databázového dotazu jsou z cache výsledků dotazů vráceny primární klíče objektů. Tyto objekty jsou instanciovány a naplněny hodnotami z cache atributů. Jejich kolekce jsou potom získány z cache relací a instanciovány a naplněny taktéž z cache atributů. Není přitom volán žádný dotaz do databáze. Bez použití vyrovnávací paměti by byl volán minimálně jeden dotaz a pravděpodobně také další dotazy na získání objektů z kolekcí, pokud by k nim bylo přistupováno. Aby bylo využití vyrovnávací paměti efektivní, je potřeba vše správně nastavit a dodržovat určitá pravidla jejího použití.

Nastavení second-level cache

Second-level cache je potřeba nejprve globálně zapnout v nastavení rozhraní *SessionFactory*. V systému se toto nastavení nachází v konfiguračním souboru *applicationContext-hibernate.xml* a je blíže popsáno již v kapitole 5.3.2.

Cache relací je nutné zapnout u každého perzistentního objektu zvlášť pomocí anotace.

```
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE, region = "userCache")
public class User extends AbstractEntity implements UserDetails {
```

Atribut `usage` určuje způsob použití cache. V tomto případě udává, že objekt `User` se bude z cache číst i zapisovat. Atribut `region` určuje logický region v rámci second-level cache, do kterého se bude tento objekt ukládat. V aplikaci je možné vytvořit teoreticky nekonečné množství regionů, ale každý se musí definovat v konfiguračním souboru *ehcache.xml*. U každého regionu lze nastavit maximální počet položek, který bude uchovávat, trvanlivost každé položky nebo zda se má obsah regionu uložit na disk při vypnutí aplikace a po startu opět načíst do paměti.

Cache relací se nastavuje podobným způsobem v místě, kde se také definuje objektově relační mapování. V této aplikaci je to tedy nad get metodami dané relace.

```
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE, region = "userCache")
public Set<Role> getRoles() {
```

Cache dotazů se potom nastavuje u každého dotazu v rozhraní *Query*. V následující ukázce je zobrazen databázový dotaz v jazyce HQL pro získání uživatele podle uživatelského jména se zapnutou cache dotazů.

```
StringBuffer hql = new StringBuffer();
hql.append("SELECT user FROM User user ");
hql.append("WHERE user.username = :username AND user.enabled = true ");
Query query = session.createQuery(hql.toString());
query.setString("username", username);
query.setMaxResults(1);
query.setCacheable(true);
query.setCacheRegion("userCache");
return query.uniqueResult();
```

Zde se nabízí otázka, jak se tato cache zachová při aktualizaci daného záznamu uživatele, aby nevracela neaktuální data. Na tuto otázku odpovím popisem celého procesu volání databázového dotazu se zapnutou cache.

Nejprve se sestaví klíč do cache, který je tvořen samotným dotazem a jeho parametry. Poté se prohledá cache, zda tento klíč neobsahuje. Pokud ho neobsahuje, tak se spustí daný dotaz do databáze a jeho výsledek se do cache pod tímto klíčem uloží. Pokud klíč obsahuje, tak se zkontroluje, zda nebyly tabulky, nad kterými dotaz pracuje, v poslední době změněny. Změny v tabulkách zaznamenává tzv. *UpdateTimestampsCache*. Kdykoli je některá tabulka aktualizována, zapíše se časová známka této poslední aktualizace do *UpdateTimestampsCache*. Každý záznam v cache má také svoji časovou známku, která určuje čas jeho vložení. Tato časová známka nalezeného záznamu v cache se tedy porovná s časovými známkami všech tabulek v *UpdateTimestampsCache* a pouze pokud jsou časové známky všech tabulek starší než známka nalezeného záznamu, tak je tento získán z cache. V opačném případě se spustí dotaz do databáze a výsledek se opět uloží do cache.

Pravidla pro efektivní využití second-level cache

Hibernate second-level cache může velmi podstatně zvýšit rychlost odezvy aplikace a velmi odlehčit zatížení databázového systému. Při špatném použití však nemusí být žádné zvýšení výkonu patrné nebo může dokonce dojít k jeho zhoršení nebo zaplnění operační paměti. Pro maximální efektivitu je nutné dodržovat následující pravidla:

- Nepoužívat pro dotazy s často se měnícími parametry – počet výsledků, které se získají z cache (tzv. hit rate), potom bude velmi nízký, operace spojené s prohledáváním cache budou zbytečně zpomalovat a v paměti cache se bude udržovat zbytečně mnoho záznamů.
- Nepoužívat pro dotazy nad tabulkami, které se často aktualizují – při každé aktualizaci budou veškeré záznamy v cache zneplatněny a stejně se bude muset provést dotaz do databáze. Z tohoto důvodu není použita pro dotazy získávající telefonní kontakt pro další hovor operátora pracujícího na PC, jejichž optimalizace je popsána v kapitole 5.7.1.
- Nepoužívat pro dotazy, které pracují s aktuálním časem – např. dotaz s databázovou funkcí pro získání aktuálního data *now()* nám bude vracet výsledek z cache, který již nemusí být aktuální. V klíči do cache totiž bude řetězec *now()* a nikoliv aktuální datum. Na druhou stranu pokud bude v parametru dotazu zadán přesný čas, tak bude téměř nulová hit rate, protože při každém dalším volání dotazu bude tento čas jiný.

- Dotazy nad tabulkami, jejichž obsah se nikdy nemění (např. některé číselníky), by měly využívat cache v režimu pouze pro čtení – práce s cache se urychlí o operace spojené se zjišťováním časových známek tabulek a záznamů.
- V parametrech HQL dotazů by se neměly vyskytovat celé objekty, ale raději pouze jejich identifikátory – reprezentace objektů zabírá hodně paměťového místa v klíčích do cache.

Další možností, jak optimálně vyladit nastavení second-level cache, je její rozdělení do regionů. Pokud není zadán žádný region, tak se všechny položky ukládají do jednoho implicitního regionu. Definováním nového regionu pro některé objekty a databázové dotazy však můžeme zaručit, že pro ně bude vyhrazeno místo v paměti pro předem určený počet záznamů, a že toto místo nebude zabráno jinými méně často používanými dotazy. Zároveň je možné každému regionu nastavit jinou platnost jeho položek. Nebo nastavit maximální počet záznamů, které může uchovávat, aby případně nezabíral příliš mnoho místa v operační paměti. V aplikaci jsem vytvořil celkem tři regiony s různým počtem maximálních položek v paměti. První region slouží pro uchovávání dat o uživatelích, druhý o projektech a třetí o telefonních hovorech. Konfigurace regionů se nachází v souboru *ehcache.xml* v adresáři *src/main/resources*. Celá ukázka tohoto souboru je v příloze 3. Význam všech atributů v nastavení jednotlivých regionů je možné nalézt v dokumentaci projektu Ehcache [33].

6 Závěr

Cílem této diplomové práce bylo navržení a vytvoření systému pro společnost zabývající se telefonováním, který co nejvíce usnadní a automatizuje práci jejích zaměstnanců. Systém splňuje požadavky běžných firemních informačních systémů a poskytuje správu zaměstnanců, klientů a jejich projektů. Navíc je propojen s telefonní ústřednou, ze které získává informace o všech provedených hovorech včetně jejich hlasových záznamů. První část práce se věnovala definici požadavků a návrhu tohoto systému. Druhá část představila použité technologie se zdůvodněním jejich výběru. Následně byl podrobně popsán celý postup implementace, který zahrnoval konfiguraci a integraci jednotlivých technologií, implementaci datové vrstvy, servisní vrstvy, prezentační vrstvy a nakonec také propojení s telefonní ústřednou Asterisk a optimalizaci aplikace.

Oproti zadání aplikace počítá s tím, že klientem může být také cizí subjekt a ne pouze oddělení či pobočka dané společnosti. Dále byla oproti zadání navíc provedena optimalizace nejkritičtější části aplikace a byl ukázán postup, jaký by se mohl aplikovat při kompletní optimalizaci celého systému.

Pokud by měl být systém reálně nasazen, tak bych jistě nadále pokračoval v jeho vývoji, jelikož poskytuje dost prostoru pro další rozšíření a vylepšení. Prvním rozšířením by jistě byla implementace požadavku č. 15 uvedeného v kapitole 3.5. Tento požadavek nebyl součástí zadání, a proto nebyl z časových důvodů realizován. Dalším možným rozšířením by mohlo být automatické zpracování vytištěných formulářů hovoru operátorů pracujících bez PC. Automatického zpracování by mohlo být dosaženo použitím scanneru, který by na základě značek ohraničujících místa pro jednotlivé odpovědi tyto ukládal do systému. Dalším krokem by pak mohlo být zpracování ručně psaného textu těchto odpovědí do digitální podoby. Výsledkem by bylo ušetření velkého množství práce manažerům jednotlivých projektů. Otázkou ovšem je, zda by se vyplatilo společnosti investovat do takovýchto technologií, nebo zda by již nebylo lepší vybavit všechny operátory osobním počítačem. Posledním důležitým rozšířením je ošetření některých nedořešených situací zobrazených v tabulce 5.1 v kapitole 5.6.2. Systém by měl zejména automaticky rozpoznávat položení hovoru ze strany volaného nebo přerušení spojení s telefonní ústřednou a tento fakt oznamovat operátorovi a případně ho také ukládat k informacím o daném hovoru. Tohoto chování by mohlo být dosaženo pomocí technologie Reverse AJAX, kterou poskytuje nástroj Direct Web Remoting. Použitím metody Polling by se prohlížeč v pravidelných intervalech dotazoval serveru, zda je vše v pořádku a spojení nebylo přerušeno.

Největším přínosem této práce byla pro mě možnost vyzkoušet si práci s celou řadou moderních technologií v oblasti vývoje Java EE aplikací. Ukázalo se, že volba aplikačního rámce je velmi vhodná pro tento typ aplikace, jelikož poskytuje dobrou podporu pro integraci dalších nástrojů. Rámec Hibernate se zase předvedl jako vhodný nástroj pro implementaci datové vrstvy díky pohodlné práci s relační databází na objektové úrovni a díky širokým možnostem v ladění výkonu. Zároveň jsem se seznámil s telekomunikačním systémem Asterisk, který poskytuje velké množství služeb v oblasti telefonování, z nichž některé jsem využil v této práci. Vyzkoušel jsem si také, jakým způsobem lze obejít technologická omezení v komunikaci mezi serverem a klientem u webové aplikace.

Literatura

[1] *Unified Modeling Language* [online]. 1997-2011 [cit. 2011-05-16]. Dostupné z WWW: <http://www.uml.org/>.

[2] *Enterprise Architect* [online]. 2000-2011 [cit. 2011-05-16]. Dostupné z WWW: <http://www.sparxsystems.com.au/>.

[3] *Spring Framework* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://www.springsource.org/about>.

[4] JOHNSON, R.; HOELLER, J. *Expert One-on-One: J2EE Development Without EJB.* Indianapolis : Willey Publishing, Inc., 2004. 535 s.

[5] *SpringSource* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://www.springsource.com/>.

[6] *Hibernate - Relational Persistence for Java and .NET* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://www.hibernate.org/>.

[7] *PostgreSQL* [online]. 1996-2011 [cit. 2011-05-16]. Dostupné z WWW: <http://www.postgresql.org/>.

[8] *JQuery* [online]. 2010 [cit. 2011-05-16]. Dostupné z WWW: <http://jquery.com/>.

[9] *DWR - Easy Ajax for JAVA* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://directwebremoting.org/dwr/index.html>.

[10] *Asterisk - The Open Source Telephony Projects* [online]. 2010 [cit. 2011-05-16]. Dostupné z WWW: <http://www.asterisk.org/>.

[11] *Apache Tomcat* [online]. 1999-2011 [cit. 2011-05-16]. Dostupné z WWW: <http://tomcat.apache.org/>.

[12] *Apache Maven* [online]. 2002-2011 [cit. 2011-05-16]. Dostupné z WWW: <http://maven.apache.org/>.

[13] *Apache Ant* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://ant.apache.org/>.

[14] HUVAR, M. *Vnitřní informační systém malé firmy*. Brno, 2008. 35 s. Bakalářská práce. FIT VUT.

[15] *Eclipse IDE* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://www.eclipse.org/>.

[16] FOWLER, M. *Inversion of Control Containers and the Dependency Injection pattern* [online]. 2004-01-23 [cit. 2011-05-16]. Dostupné z WWW: <http://martinfowler.com/articles/injection.html>.

[17] *FreeMarker* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://freemarker.sourceforge.net/>.

[18] *XHTMLRenderer - Flying Saucer* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://xhtmlrenderer.java.net/>.

[19] BAUER, Ch.; KING, G. *Hibernate in Action.* Greenwich : Manning Publications Co, 2005. 395 s.

[20] *C3p0:JDBC DataSources/Resource Pools* [online]. 2011 [cit. 2011-05-16]. Dostupné z WWW: <http://sourceforge.net/projects/c3p0/>.

[21] *Joda-Time - Java date and time API* [online]. 2002-2011 [cit. 2011-05-16]. Dostupné z WWW: <http://joda-time.sourceforge.net/>.

[22] *Hibernate Tools* [online]. 2011 [cit. 2011-05-17]. Dostupné z WWW: <http://www.hibernate.org/subprojects/tools.html>.

[23] MACHACEK, J.; VUKOTIC, A.; CHAKRABORTY, A. *Pro Spring 2.5*. [s.l.] : Apress, 2008. Transaction Management, s. 575-610.

[24] *Aspect Oriented Programming with Spring* [online]. 2011 [cit. 2011-05-17]. Dostupné z WWW: <http://static.springsource.org/spring/docs/current/reference/aop.html>.

[25] *Spring Transaction Management* [online]. 2011 [cit. 2011-05-17]. Dostupné z WWW: <http://static.springsource.org/spring/docs/current/reference/transaction.html#tx-decl-explained>.

[26] *Spring DispatcherServlet* [online]. 2011 [cit. 2011-05-17]. Dostupné z WWW: <http://static.springsource.org/spring/docs/current/reference/mvc.html#mvc-servlet>.

[27] *AsteriskNOW* [online]. 2010 [cit. 2011-05-17]. Dostupné z WWW: <http://www.asterisk.org/asterisknow/>.

[28] *FreeSSHd and freeFTPd* [online]. 2008-2009 [cit. 2011-05-17]. Dostupné z WWW: <http://www.freesshd.com/>.

[29] *Sshj - SSHv2 library for Java* [online]. 2011 [cit. 2011-05-17]. Dostupné z WWW: <https://github.com/shikhar/sshj>.

[30] *Pjsua Manual Page* [online]. 2007 [cit. 2011-05-17]. Dostupné z WWW: <http://www.pjsip.org/pjsua.htm>.

[31] *Apache JMeter* [online]. 1999-2011 [cit. 2011-05-17]. Dostupné z WWW: <http://jakarta.apache.org/jmeter/>.

[32] *Ehcache* [online]. 2003-2010 [cit. 2011-05-17]. Dostupné z WWW: <http://ehcache.org/>.

[33] *Ehcache CacheConfiguration JavaDoc* [online]. 2011 [cit. 2011-05-17]. Dostupné z WWW: <http://ehcache.org/apidocs/net/sf/ehcache/config/CacheConfiguration.html>.

Seznam příloh

Příloha 1: Konfigurační soubor

V této příloze se nachází základní konfigurační soubor prostředí *config.properties*, který slouží k nastavení připojení k databázím a telefonní ústředně, nastavení složky pro ukládání souborů a nahrávání záznamů hovorů. Nachází se ve složce *src/main/config/*.

```
## pripojeni k PostgreSQL databazi systemu
################################################################
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
hibernate.connection.driver_class=org.postgresql.Driver
hibernate.connection.url=jdbc:postgresql://localhost:5432/telesystem
hibernate.connection.username=telesystem
hibernate.connection.password=telesystem

hibernate.show.sql=false

## slozka na ukladani souboru
files_save_folder=/var/lib/tomcat6/files/

## pripojeni k Asterisk databazi a k ssh na PC operatora
################################################################
asterisk_connection_ip=
telephone_ssh_username=asterisk
telephone_ssh_password=asterisk

telephone_test_sip_id=200
telephone_operator_ip=192.168.56.102

## pripojeni k CDR databazi telefonni ustredny
################################################################
cdr.connection.dialect=org.hibernate.dialect.MySQLDialect
cdr.connection.driver_class=com.mysql.jdbc.Driver
cdr.connection.url=jdbc:mysql://localhost:3306/asteriskcdrdb
cdr.connection.username=asterisk
cdr.connection.password=asterisk

## slozka s nahravkami hovoru
cdr_records_folder=/var/spool/asterisk/monitor/
```

Příloha 2: Konfigurace Hibernate

Tato příloha zobrazuje ukázku z konfiguračního souboru datové vrstvy rámce Hibernate. Obsahuje nastavení rozhraní *SessionFactory* a datového zdroje využívajícího udržovaní spojení pomocí knihovny c3p0. Celý soubor se v aplikaci nachází ve složce *src/main/webapp/WEB-INF/spring/applicationContext-hibernate.xml*.

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
    <property name="configLocation" value="classpath:hibernate.cfg.xml" />
    <property name="hibernateProperties">
        <props>
             <prop key="hibernate.dialect">${hibernate.dialect}</prop>
             <prop key="hibernate.query.factory_class">
                 org.hibernate.hql.classic.ClassicQueryTranslatorFactory
             </prop>
             <prop key="hibernate.bytecode.use_reflection_optimizer">false</prop>
             <prop key="hibernate.show_sql">${hibernate.show.sql}</prop>
             <prop key="hibernate.use_sql_comments">${hibernate.show.sql}</prop>
             <prop key="hibernate.jdbc.use_scrollable_resultset">true</prop>
             <prop key="hibernate.jdbc.batch_size">20</prop>
             <prop key="hibernate.format_sql">true</prop>
             <prop key="hibernate.hibernate.generate_statistics">true</prop>
             <prop key="hibernate.cache.use_structured_entries">true</prop>
             <prop key="hibernate.cache.use_second_level_cache">true</prop>
             <prop key="hibernate.cache.provider_class">
                 net.sf.ehcache.hibernate.EhCacheProvider
             </prop>
             <prop key="hibernate.cache.use_query_cache">true</prop>
        </props>
    </property>
    <property name="dataSource" ref="c3p0DataSource"/>
</bean>

<bean id="c3p0DataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
destroy-method="close">
    <property name="driverClass" value="${hibernate.connection.driver_class}" />
    <property name="jdbcUrl" value="${hibernate.connection.url}" />
    <property name="properties">
        <props>
            <prop key="user">${hibernate.connection.username}</prop>
           <prop key="password">${hibernate.connection.password}</prop>
        </props>
    </property>
    <property name="initialPoolSize" value="15" />
    <property name="minPoolSize" value="15" />
    <property name="maxPoolSize" value="60" />
    <property name="maxIdleTime" value="300" />
    <property name="maxStatements" value="50" />
    <property name="idleConnectionTestPeriod" value="150" />
</bean>
```

Příloha 3: Konfigurační soubor Hibernate cache

Poslední příloha obsahuje konfigurační soubor pro vyrovnávací paměť rámce Hibernate. Vyrovnávací paměť je implementována knihovnou Ehcache a tento soubor tedy obsahuje specifická nastavení pro tuto implementaci. V aplikaci se nachází ve složce *src/main/resources/*.

```
<ehcache>
    <!-- cesta na disku, kde bude ulozena diskova cache -->
    <diskStore path="/var/lib/tomcat6/ehcache/"/>

    <defaultCache
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="true"
        diskPersistent="false"
        diskExpiryThreadIntervalSeconds="120"
        />

    <!-- Cache uchovavajici casove znamky o zmenach v tabulkach. Musi mit delsi
    zivotnost nez vsechny cache, nejlepe nekonecnou. -->
    <cache
       name="org.hibernate.cache.UpdateTimestampsCache"
       maxElementsInMemory="5000"
       eternal="true"
       overflowToDisk="true"/>

    <!-- Cache uchovavajici vse okolo uzivatelu. -->
    <cache name="userCache"
        maxElementsInMemory="1000"
        eternal="false"
        overflowToDisk="false"
        timeToLiveSeconds="864000"
        diskPersistent="true"
        />

    <!-- Cache uchovavajici vse okolo projektu. -->
    <cache name="projectCache"
        maxElementsInMemory="5000"
        eternal="false"
        overflowToDisk="false"
        timeToLiveSeconds="864000"
        diskPersistent="true"
        />

    <!-- Cache uchovavajici vse okolo telefonnich hovoru. -->
    <cache name="phonecallCache"
        maxElementsInMemory="10000"
        eternal="false"
        overflowToDisk="false"
        timeToLiveSeconds="864000"
        diskPersistent="true"
        />

</ehcache>
```